Haaga-Helia
ammattikorkeakoulu Oy

PowerShell: Windows Server 2016 ja Active Directory hallinta

Teemu Keski-Simonen

Opinnäytetyö
Tietojenkäsittelyn koulutusohjelma
2018

Tiivistelmä

Haaga-Helia ammattikorkeakoulu Oy

Tekijä(t)
Teemu Keski-Simonen

Koulutusohjelma
Tietojenkäsittelyn koulutusohjelma

Raportin/Opinnäytetyön nimi
PowerShell: Windows Server 2016 ja Active Directory hallinta

Sivu- ja liitesivumäärä
44

Opinnäytetyö käsittelee Windows PowerShellin käyttöä Windows Server 2016 ja Active Directoryn hallinnassa. Opinnäytetyö esittelee PowerShellin ominaisuuksia, käyttölogiikkaa sekä komentoja. Aihe on rajattu Active Directoryn ja Windows Server 2016 hallintaan.

PowerShell on Microsoftin vuonna 2006 julkaisema interaktiivinen komentokehote ja komentosarjakieli. PowerShellillä voi hallita Windows käyttöjärjestelmää sekä Microsoftin muita, etenkin palvelinympäristöön julkaisemia ohjelmistoja.

Teoriaosuudessa käydään läpi Windows palvelinkäyttöjärjestelmää, aktiivihakemiston toimintaa sekä PowerShellin toimintalogiikkaa ja ominaisuuksia.

Toiminnallisessa osuudessa tutustutaan runsain käytännön esimerkein PowerShell komentoihin ja lopuksi luodaan käytännöllinen komentosarjaesimerkki sekä pohditaan PowerShellin hyötyjä verrattaessa Windowsin graafisiin työkaluihin.

Opinnäytetyöprojektin lopputuloksena syntyi käytännönläheinen opas PowerShellin käyttömahdollisuuksiin Windows-palvelinympäristössä. Opinnäytetyössä käsiteltyjä perusteita voidaan soveltaa myös muidenkin sovellusten hallintaan PowerShellillä.

Asiasanat
PowerShell, Active Directory, aktiivihakemisto, Windows Server 2016

Sisällys

1 Johdanto

Opinnäytetyö käsittelee Microsoftin kehittämää, jo vuonna 2006 julkaistua äärimmäisen tehokasta ja interaktiivista komentokehotetta ja skriptikieltä nimeltä PowerShell. PowerShell on kuin moderni versio Linuxin komentokehotteesta Bashista ja esimerkiksi Python-ohjelmointikielestä samassa paketissa. On sitten tarve nopeasti luoda jokin yksinkertainen skripti nopeasti jonkin ongelman ratkaisemiseen tai vastaavasti tuotantokelpoinen, jopa vuosia käytettävä työkalu. Vastaus tarpeeseen on PowerShell. PowerShell tarjoaa erilaisille käyttäjälle ja erilaisiin tilanteisiin oivan tavan luoda omia työkaluja ja vastauksia ylläpitäjän pulmiin. Siksi on tärkeää, etenkin Windows ekosysteemissä työskenteleville ylläpitäjille, tietää mikä on PowerShell ja miten sitä käytetään.

PowerShellin hyödyt ovat kiistattomia: sitä voidaan käyttää todella monien eri teknologioiden kanssa esimerkiksi websivujen, virtuaalikoneiden, Microsoftin tuotteiden, XML, CVS ja niin edelleen. Objektipohjaisuutensa takia PowerShell tarjoaa sisäänrakennettuja ominaisuuksia tiedon käsittelyyn ja lähes kaikki objektit pystytään ohjamaan putken läpi toisiin komentoihin, mikä on erittäin tehokasta. Microsoft on myös panostanut ja tulee tulevaisuudessakin panostamaan PowerShelliin suuresti, se ei ole menossa mihinkään – päinvastoin yhä useamman tuotteen hallinta onnistuu PowerShellillä ja joidenkin ominaisuuksien hallinta ei onnistu esimerkiksi graafisen käyttöliittymän kautta, joten joskus on pakko avata komentokehote. PowerShell tarjoaa myös mahdollisuuden kehittää sillä tehdyille sovelluksille graafisen käyttöliittymän. Automatisoinnin tarve on suurta etenkin palvelinympäristöissä ja PowerShell tarjoaa tähänkin ratkaisuja. Myös muiden valmistajien tuki PowerShellille kasvaa koko ajan. (Jones & Frame 2016, 5-9)

Opinnäytetyössä esitellään aluksi käsiteltävät ohjelmistot ja avataan niihin liittyviä termejä ja esitellään ominaisuuksia sekä lopuksi pureudutaan hieman tarkemmin käytännön sovellutuksiin. Tämän opinnäytetyön tärkein tavoite on koota laadukkaista lähteistä ja mahdollisimman tuoreista julkaisuista tietoa PowerShellin käytöstä ja ominaisuuksista ja ehkä myös innostaa lukija käyttämään PowerShelliä päivittäisissä ylläpitotoimissaan. Opinnäytetyö on rajattu käsittelemään aktiivihakemiston ja Windows Server 2016 palvelinkäyttöjärjestelmän ylläpitotehtäviä sekä esittelemään PowerShellin perusominaisuuksia. Hallinta-osioissa on keskitytty yleisempiin ja mahdollisesti usein toistuviin ylläpitotehtäviin. Ylläpitäjän on mahdollista säästää selvää aikaa tekemällä asioita PowerShellillä kuin perinteisesti graafisen käyttöliittymän kautta.

Käytännön esimerkit ja kuvankaappaukset on tehty kahdella samassa toimialueessa olevalla Windows Server 2016 palvelimella virtuaaliympäristössä. Toiseen palvelimeen on

asennettu Core-versio ja toiseen täysi graafinen käyttöliittymä. Projektin toimialueelle on myös luotu testikäyttäjätunnuksia ja ryhmiä. Käytetty PowerShell versio on 5.1.

2 Windows Server 2016

Windows Server 2016 on huhtikuussa 2016 julkaistu Microsoftin palvelinkäyttöjärjestelmä. Käyttöjärjestelmän kehittelyssä on panostettu käyttöjärjestelmän valmiutta sulautua pilvipalveluhin, etenkin Microsoftin omaan Azureen. Suunnittelussa on myös panostettu käyttöjärjestelmän turvallisuuteen tarjoamalla eri tyyppisiä suojauksia. Käyttöjärjestelmä noudattaa Windows 10:stä tuttua ulkonäköä, mutta tarjolla on myös mahdollisuus asentaa käyttöjärjestelmä kokonaan ilman graafista käyttöliittymä, tällöin asennus on pienempi ja turvallisempi. (McCabe 2016, 1-3)

2.1 Uudet ominaisuudet

Uusia ominaisuuksia on runsaasti. Näkyvin uudistus on kuitenkin siirtyminen Windows 10 versiosta tuttuun graafiseen yleisilmeeseen. PowerShell versio 5.0 tulee esiasennettuna käyttöjärjestelmän mukana. Kaikki asetukset mitä graafisessa näkymässä tehdään, voidaan myös tehdä PowerShellin komentorivillä. Osa asetuksista on mahdollista tehdä vain PowerShellillä. Windows Defender on ensimmäistä kertaa Windows palvelinkäyttöjärjestelmässä suojaamassa haittaohjelmilta. Muita parannuksia on virtuaalikoneiden suojauksessa, uusi Nano-server -asennus sekä kevyt uudelleenkäynnistys (Soft Restart), mikä käynnistää käyttöjärjestelmän uudelleen käynnistämättä palvelinta fyysisesti uudelleen. (Krause 2016, 5-7)

2.2 Eri asennusversiot

Käyttöjärjestelmästä voidaan asentaa kolme eri versiota: Desktop Experience eli perinteinen graafinen käyttöliittymä, Core Server sekä Nano Server. Perinteinen Desktop-versio on kaikista aiemmista Windows Server versioista tuttu graafinen Windows ympäristö.

Core-asennus on ollut Windows palvelinkäyttöjärjestelmän asennusvaihtoehtona jo vuodesta 2008 lähtien. Core on täysiverinen Windows Server ilman graafista käyttöliittymää. Käyttöjärjestelmä on tällöin kompaktimpi ja se vie vähemmän muistia, kiintolevytilaa ja siinä on merkittävästi pienempi hyökkäyspinta-ala verkkorikollisille verrattuna perinteiseen Windows Desktop Experience asennukseen. Myös automaatio ja järjestelmän skaalautuvuus esimerkiksi pilveen tukee graafisen käyttöliittymän tarpeettomuutta. Järjestelmän hallinta hoidetaan merkkipohjaisesti paikallisesti tai etänä esimerkiksi PowerShellillä. Ser-

ver Core on oletusasennusvaihtoehto Windows Server 2016 asennusohjelmassa. Mahdollisuus asentaa graafinen käyttöliittymä jälkikäteen, tai sen poistaminen, on otettu pois käytöstä Windows Server 2016 julkaisussa. (Krause 2016, 237-240)

Nano Server on täysin uusi pelkästään Windows Server 2016:lle kehitetty asennusmuoto. Nano Server on monella tapaa samanlainen kuin Core Server, mutta sen tarkoituksena on olla pienempi kooltaan ja kuten Core-versiossakin hyökkäyspinta-alan pienetessä turvallisuus lisääntyy. Nano on 20 kertaa pienempi kuin Core Server, noin 90% pienempi kiintolevytarve sekä merkittävästi vähemmän tarvetta uudelleenkäynnistyksille. Nano Server tukee vain 64 bittisiä sovelluksia ja se on nopea käynnistymään. (McCabe 2016, 89-91)

Nano Server suunniteltiin hallittavaksi täysin etänä, mutta nykyään sitä on mahdollista myös hallita paikallisesti. Nano Server kehittyy koko ajan ja tällä hetkellä siihen voidaan asentaa muun muassa Hyper-V, tiedostopalvelin, DNS-palvelin, IIS-webpalvelin sekä Cloud Application Server -roolit. Nano roolia ei voi tällä hetkellä vielä asentaa suoraan Windows Asennusohjelmalla. Asennus tehdään luomalla virtuaalinen VHD-levykuva asennusmedian mukana tulevilla työkaluilla, mikä voidaan muuttaa virtuaalikoneeksi. Microsoft on myös valmistanut työkalun millä asennus voidaan tehdä USB-tikun avulla. (Krause 2016, 254-260)

2.3 Roolit ja toiminnot

Jotta palvelimella voisi tehdä jotain hyödyllistä, tulee sille antaa jokin rooli. Asentamalla jonkin roolin määritellään palvelimen tehtävä verkossa. Asennettavia rooleja voivat olla esimerkiksi: Aktiivihakemisto, DHCP-palvelin, DNS-palvelin, tiedostopalvelin, tulostinpalvelin tai webpalvelin. Toiminnot ovat taas palvelimelle asennettavia tai aktivoitavia ominaisuuksia esimerkiksi Bitlocker-salaus tai PowerShell. Kaikkia toimintoja ei ole oletuksena asennettu tai laitettu päälle, koska niitä ei aina tarvita. (Krause 2016, 35-36, 253)

3 Active Directory

Aktiivihakemisto (Active Directory) on Microsoftin suunnittelema Windowsin päälle rakennettu verkkokäyttöjärjestelmä. Aktiivihakemisto mahdollistaa laajojen tietokokonaisuuksien keskitetyn hallinnan ja jakelun. Aktiivihakemisto muodostuu metsistä, puista sekä toimialueista.

Aktiivihakemistoon voidaan lisätä käyttäjiä, ryhmiä, tietokoneita, tulostimia, ohjelmia sekä palveluita ja nämä kaikki voidaan tarjota saataville koko hallittavan organisaation käyttäjille. Tiedon ja laitteiden näkyvyyttä voidaan säädellä halutulla tavalla esimerkiksi organisaatioyksiköiden (OU) avulla. (Desmond & Richards, ym. 2013, 1)

3.1 Active Directory Domain Services

AD DS on hakemistopalvelu, mikä tarjoaa tapoja tallentaa hakemiston tietoja kuten käyttäjätunnuksien nimiä ja salasanoja. AD DS on esitelty ensimmäisen kerran Windows 2000 palvelin versiossa Windows NT 4.0 toimialueiden korvaajana. Yleisesti alalla tunnustettu ja hyväksytty AD DS on osoittautunut luotettavaksi, skaalautuvaksi ja tehokkaaksi järjestelmäksi. Skaalautuvuutensa takia AD DS sopii mainiosti niin pienistä isoihin toimistoihin ja jopa monikansallisiin yrityksiin. Uusimmassa versiossa on uusina toimintoina muun muassa mahdollisuus luoda virtuaalisia ohjauskoneita sekä palauttaa jo poistettuja objekteja. (Morimoto & Noel, ym. 2017, 134-138)

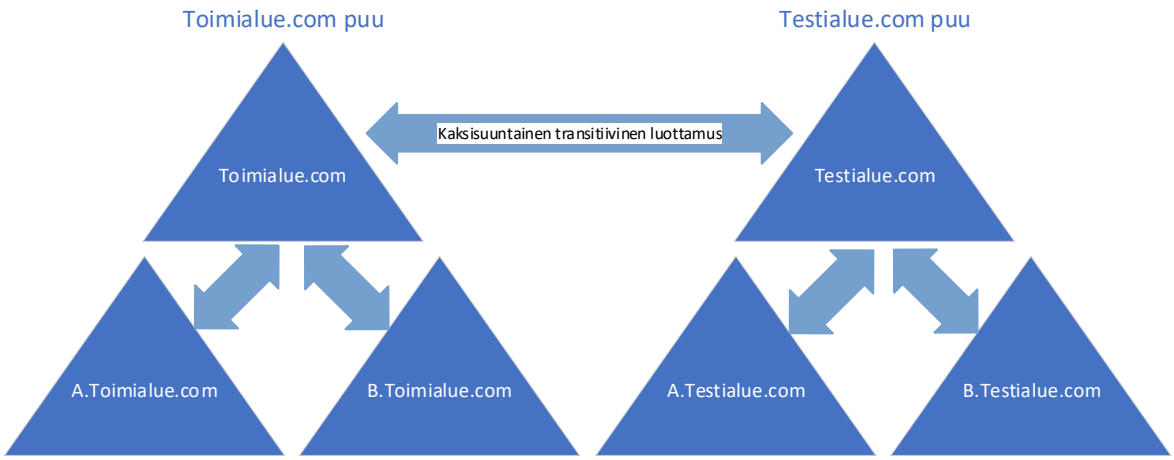
3.2 Metsät

Aktiivihakemiston rakenteessa puut ovat kokoelma toimialueita näin ollen metsä on yhden tai useamman puun kokoelma. Metsä syntyy samalla kun ensimmäinen toimialue luodaan. Puut jakavat yhteisen skeeman sekä määritysluettelon minkä lisäksi puut ovat yhdistetty toisiinsa kaksisuuntaisella transitiivisella luottamuksella. Tarvittaessa metsien välille on mahdollista luoda yhden- tai kahdensuuntaisia transitiivisia luottamuksia. (Desmond & Richards, ym. 2013, 11-12)

3.3 Puut

Puut ovat toisiinsa hierarkkisella mallilla toisiinsa liitettyjä toimialueita. Puut helpottavat resurssien hallintaa, koska kaikki puurakenteen toimialueet luottavat automaattisesti toisiinsa, tapahtumaa kutsutaan *transitiiviseksi luottamukseksi*. Transitiivinen luottamus voi olla joko yhden- tai kahdensuuntaista.

Kuva 1 esittää, että transitiivisen luottamuksen toimesta, A.Toimialue.com kuuluvalla käyttäjällä on mahdollista päästä käsiksi B.Testialue.com toimialueen resursseihin. Järjestelmävalvoja määrittää pääsyoikeuksia puolin ja toisin tarpeen mukaan, mutta kaikki oikeudet eivät ole oletusarvoisesti päällä. (Desmond & Richards, ym. 2013, 10-11)



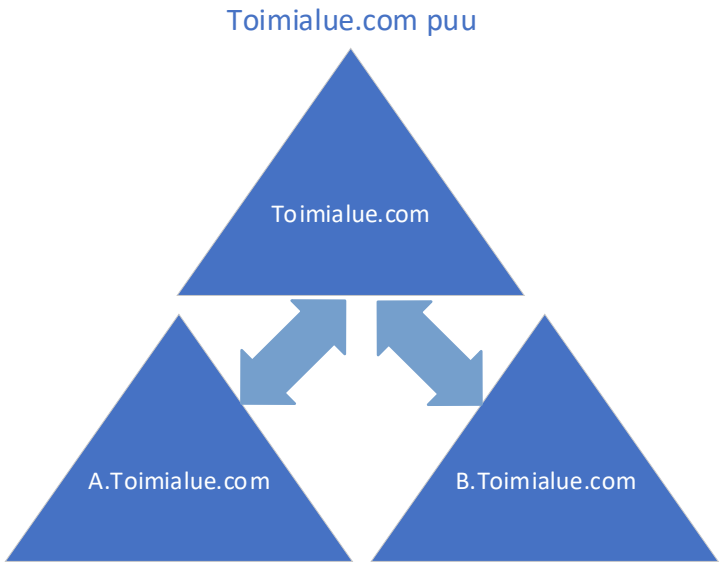
Kuva 1: Kaksisuuntainen transitiivinen luottamus

3.4 Toimialue

Aktiivihakemiston looginen rakenne perustuu toimialueisiin (Domain). Yhtä toimialuetta voidaan kuvata kolmiolla, kuten kuvassa 2. Käyttäjät ja tietokoneet ovat tallennettu toimialueeseen ja niitä hallitaan toimialueen rajojen sisällä. Toimialueet ovat tavallaan ylläpidon näkökulmasta katsottuna virtuaalisia, ei fyysiseen sijaintiin sidottuja, turvallisuusrajoja erilaisille objekteille. (Morimoto & Noel, ym. 2017)

Toimialue rakentuu seuraavista komponenteista:

- Hierarkkinen järjestelmä säiliöitä ja objekteja.
- Toimialueella uniikki DNS nimi.
- Palvelu, mikä autentikoi ja valtuuttaa pääsyn resursseihin käyttäjätunnuksilla tai luetettujen toimialueiden väleillä.
- Käyttäjille tai laitteille määritellyt käytännöt (Policy). (Desmond & Richards, ym. 2013, 9)



Kuva 2: Kolme toimialuetta muodostavat puun

3.5 Kaava (Schema)

Aktiivihakemistossa skeema eli kaava on kokoelma määrityksiä kaikille hakemistossa oleville objektityypeille ja niihin liittyville ominaisuuksille. Kaava määrittää tavan miten kaikki käyttäjät, tietokoneet ja muut objektit on säilötty ja määritelty aktiivihakemistoon. Kaava on hakemiston perusmäärittely ja keskeinen toimialueiden toiminnan kannalta. Muutokset kaavaan vaikuttavat koko aktiivihakemistoon. (Morimoto & Noel, ym. 2017, 140-141)

3.6 Ohjauskone (Domain Controller)

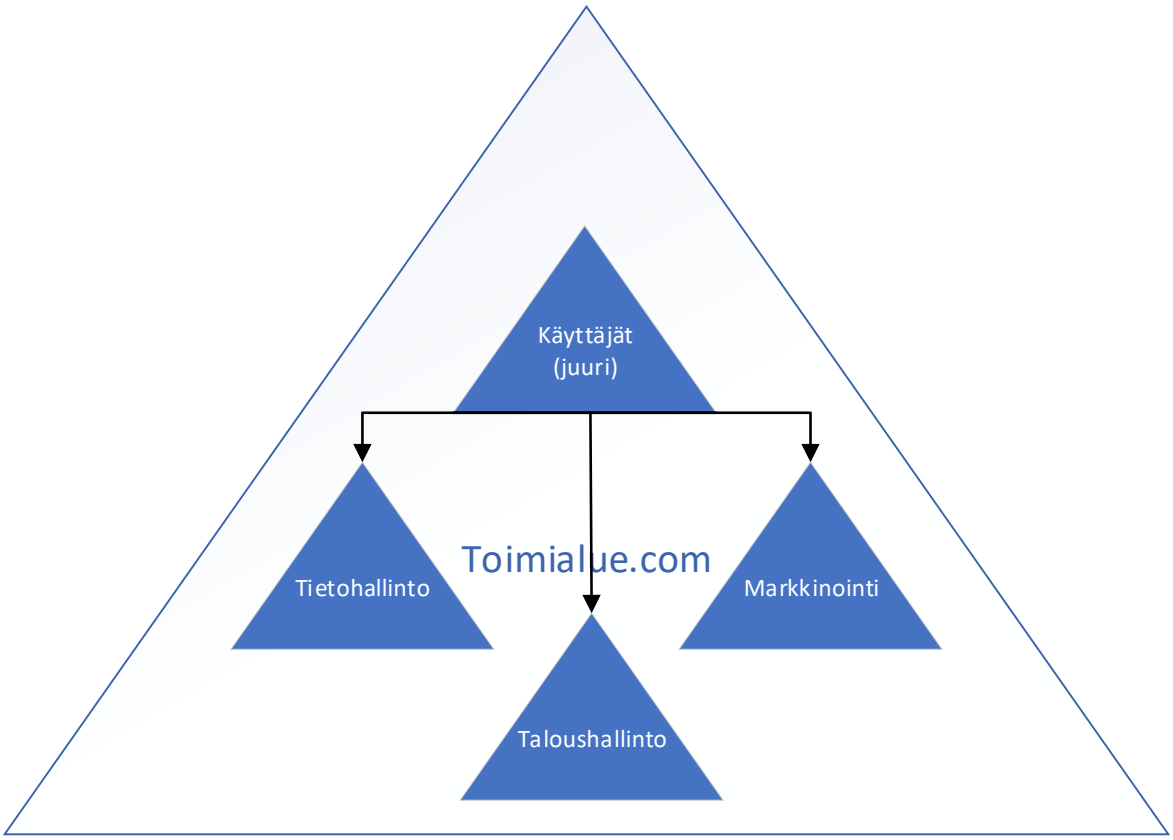
Ohjauskone on nimensä mukaisesti palvelin, mikä on vastuussa yhden toimialueen ylläpitämisestä tarjoten asiakkaille todentamista ja hakemistopalveluita. Ohjauskone voi olla myös tarpeen vaatiessa ylikirjoitussuojattu, tämä on tarpeellista silloin kuin palvelimen fyysinen sijainti ei ole tarpeeksi suojainen. (Svidergol & Allen 2013, 55)

Ohjauskone ei voi siis olla vastuussa kuin yhdestä toimialueesta, mutta ohjauskoneella voidaan tarvittaessa säilyttää kirjoitussuojattuja kopioita muista toimialueista, jos palvelin on määritelty yleisen luettelon (Global Catalog) palvelimeksi. Kaikki metsän ohjauskoneet kuitenkin säilyttävät kopioita muun muassa metsän määrityksistä sekä kaavaosioista (Schema Naming Contexts). (Desmond & Richards, ym. 2013, 9)

3.7 Organisaatioyksiköt (OU)

Organisaatioyksiköillä tarkoitetaan säiliöitä, mitkä säilövät hakemiston tietoja. Käytännössä organisaatioyksiköt ovat siis tapoja organisoida käyttäjiä, tietokoneita ja muita objekteja helpommin hallittaviin ryhmiin. Toimialueella on yksi juuri organisaatioyksikkö, minkä alle muut organisaatioyksiköt luodaan. (Morimoto & Noel, ym. 2017, 183-184)

Kuvassa kolme on jaettu käyttäjät kolmeen eri organisaatioyksikköön ryhmän ”Käyttäjät” alle.



Kuva 3: Esimerkki organisaatioyksiköistä

3.8 DNS

DNS (Domain Name System) on Windows Server käyttöjärjestelmässä palvelu, mikä yleensä otetaan käyttöön toimialueen luomisen yhteydessä. DNS ei ole kuitenkaan miten-kään sidottu Windows järjestelmiin, vaan Linuxilla ja monilla muilla järjestelmillä on omat DNS-palvelunsa, mutta tässä kappaleessa keskitytään Windows Server 2016 tarjoamaan DNS-palveluun. DNS-palvelu on vastuussa verkkonimien selvityksestä ja tallentamisesta. Käytännössä DNS-palvelu muuttaa selkokieliset verkko-osoitteet numeraalisiksi IP-osoit-teiksi ja ohjaa ne myös oikeaan paikkaan. Oikein määritetyn DNS-palvelun ansiosta testi.local -osoite internetselaimeen kirjoitettuna testiympäristössä osoittaa oikeaan IP-osoitteeseen mihin web-palvelin rooli on asennettuna. (Krause 2016, 81-82)

3.9 Ryhmät

Yksinkertaisesti sanottuna ryhmät ovat kokoelma objekteja. Aktiivihakemistossa niitä käy-tetään etupäässä käyttöoikeuksien hallintaan, ryhmäkäytäntöjen jakamiseen sekä viestien lähettämiseen. On paljon yksinkertaisempaa asettaa ryhmälle käyttöoikeuksia kuin käyttä-

jille yksitellen tai lähettää monelle tarkoitettu viesti yhteen ryhmäosoitteeseen kuin jokaiselle erikseen. Aktiivihakemistossa ryhmä voi sisältää minkälaisia objekteja hyvänsä, mutta yleensä ryhmä koostuvat vain käyttäjistä, tietokoneista ja muista ryhmistä.

Jokaisella ryhmällä on näkyvyysalue (scope) ja tyyppi (type). Tyyppi on joko turvallisuus tai jakelu. Turvallisuusryhmillä jaetaan oikeuksia tai estetään pääsy Windowsin resursseihin ja jakeluryhmiä käytetään esimerkiksi sähköpostiryhmien luomiseen. Ryhmän näkyvyysalue määrittää mihin ryhmän jäsenet sijoittuvat metsässä. Näkyvyysalue on joko *universal*, *global* tai *domain local*. Universal ja domain local näkyvyysalueilla ryhmän jäsenet voivat olla osana mitä tahansa toimialuetta saman metsän sisällä, kun taas global ryhmän jäsenien tulee kuulua samaan toimialueeseen missä ryhmäkin sijaitsee. (Svidergol & Allen 2013, 247-248)

4 PowerShell

Windows PowerShell on .NET ohjelmistokomponenttikirjaston päälle rakennettu tehtäväpohjainen, että tekstipohjainen komentokehote ja komentosarjakieli. Se on kehitetty erityisesti järjestelmien ylläpitäjille ja tehokäyttäjille ylläpitotehtävien helpottamiseen sekä työkaluksi tehtävien helpompaan automatisointiin. PowerShellin suunnittelussa on ollut määränä parantaa tekstipohjaista ympäristöä sekä komentosarjojen käytön helppoutta. PowerShell sisältää sisäänrakennetun, internetistä päivittyvän ohjeiston. Käyttäjällä on siis koko ajan mahdollisuus saada tuoreimmat ohjetiedostot päivittämällä ohjetietokantaa. Kaikkien käytössä olevien komentojen hakemisesta on myös tehty helppoa. (Microsoft 2017d)

PowerShell tukee standardeja Windows-komentoja ja sovelluksia, joten PowerShell ei pakota lopettamaan tuttujen sovellusten ja komentojen käyttämistä. Powershellin omia komentoja kutsutaan Cmdlet-komennoiksi. PowerShellissä kaikki pyörivät objektien ympärillä, tämä tarkoittaa sitä, että tavanomaisesti komento tulostaa ulos tekstipohjaista tietoa, mutta PowerShellissä työskennellään objektien kanssa ja näin tieto on moninaisempaa ja sitä pystytään helpommin käsittelemään ja myös siirtämään toisille komennoille käsiteltäväksi. (Holmes 2013, 3-4)

4.1 PowerShellin historia lyhyesti

Ensimmäinen versio julkaistiin marraskuussa vuonna 2006. Versio sisälsi noin 130 kappaletta cmdlet-komentoja yleisempiin Windows ylläpitotehtäviin. Myös vanhat Windowsin

työkalut kuten *ping* ja *ipconfig* toimivat PowerShellissä. Tuettuja Windows käyttöjärjestelmäversioita olivat XP, Server 2003 sekä Vista. (Microsoft 2008)

Versio 2.0 julkaistiin lokakuussa vuonna 2009. Päivitys sisälsi merkittäviä uudistuksia. Suurin uudistuksista oli Windows etäkäyttö PowerShellillä. Versio on myös taaksepäin yhteensopiva. Se myös paransi käytettävyyttä ja lisäsi cmdlet-komentojen määrää yli 100 kappaleella. Uusina ominaisuuksina tulivat myös integroitu komentosarjaympäristö (ISE), moduulit, komentosarjojen kansainvälistäminen, komentojen tausta-ajo ja monia muita. PowerShell versio 2.0 toimitettiin Windows 7 ja Windows Server 2008 R2 mukana ja vanhemmat Windows jakeluversiot saivat sen Service Pack-päivitysten mukana. (Microsoft 2009)

PowerShell versio 4.0 sai julkaisunsa lokakuussa 2013. Versio sisälsi parannuksia muun muassa ohjetiedostojen offline-päivittämiseen. Myöskin päivitys sisälsi runsaasti virheiden korjauksia ja nopeuden parannusta. (Microsoft 2013)

Versio 5.0 julkaistiin vuonna 2016. Uusia ominaisuuksia tuli rutkasti. PowerShellissä on mahdollista luoda ohjelmia käyttämällä luokkia, kaavamaisia syntakseja ja semantiikkaa. Uusia komentoja, yleisiä parametreja, moduuleita lisättiin. Sisällön salaaminen on myös mahdollista. Myös PowerShellin nopeutta pystyttiin parantamaan merkittävästi. (Microsoft 2017c)

4.2 Ydintoimintatapa

PowerShellin toiminta eroaa hieman muista komentokehotteista. Periaate on kuitenkin sama kuin yleisesti muissakin. Yksinkertainen komento on seuraavanlainen:

```
Komento -Parametri1 -Parametri2 Argumentti1 Argumentti2
```

Käsky alkaa komennolla joita seuraavat erilaiset parametrit. Parametrit voivat olla kytkinparametrejä (switch parameter) eli ne eivät tarvitse minkäänlaista argumenttia perään tai ne voivat olla tavallisia argumentin vaativia parametreja. PowerShellissä on myös paikkaan lukittuja parametrejä eli (positional parameter), näissä vastaava parametri päätellään argumentin sijainnin perusteella. Nämä sijainnit ovat kyseisen PowerShell komennon tekijän määrittelemiä. (Payette & Siddaway 2018, 18-20)

Esimerkiksi käskyssä *Get-Service -Name BITS* Name-parametri on positionaalinen, joten käsky *Get-Service BITS* tekee saman kuin yllä oleva ja palauttaa saman tulosteen.

Muutamia tärkeitä, kaikkien järjestelmää muuttavien komentojen kanssa toimivia niin kutsuttuja yleisiä parametrejä mitä kannattaa opetella ovat: *Confirm, Force, Verbose* sekä *Whatif*. Confirm-parametri pakottaa PowerShellin kysymään käyttäjältä ennen kuin mitään muutoksia järjestelmään tehdään. Force-parametria käytetään, kun halutaan yli kirjoittaa jotain tietoja ja muuten komento epäonnistuisi. Verbose-parametri tulostaa näytölle komennon ajamisen myötä mitä se on tehnyt, oletuksena PowerShellissä tulosteita onnistuneesta komennosta ei näytetä. Whatif-parametri nimensä mukaisesti näyttää käyttäjälle mitä komento tekisi, jos se ajetaan.

4.3 Komentotyypit

PowerShellissä voidaan käyttää neljää erilaista komentotyyppiä. Cmdlet-komennot, funktiot, komentosarjat eli skriptit, sekä natiivit Windows-ohjelmat.

4.3.1 Cmdlet-komennot

Cmdlet juontuu sanasta ja myös sanotaan: *command-let.* Kyseessä on Microsoftin kehittämä käsite nimenomaan PowerShellille. Nämä Cmdlet-komennot ovat .NET ympäristössä ohjelmoituja natiiveja PowerShell-ohjelmia. Cmdlet-komennot ovat niin kutsuttuja PowerShellin sisäänrakennettuja komentoja, vaikkakin kuka tahansa niitä voi luoda ja ajaa. Cmdlettien tekemiseen tarvitaan PowerShell ohjelmointialusta ja rajapinta (Software Development Kit). Rajapinnassa ohjelma käännetään dynaamiseksi linkkikirjastoksi (DLL), mikä ladataan käynnistyksessä PowerShelliin. Cmdlet-komentojen nimeäminen noudattaa verbi-substantiivi -käytäntöä. Verbi määrittelee toiminnan ja substantiivi tarkentaa kohteen millä toimitaan. (Payette & Siddaway 2018, 21)

4.3.2 Funktiot

Funktiot ovat komentosarjoja mitkä jäävät muistiin niin pitkäksi aikaa, kun PowerShell istunto on käynnissä.

```
Function <funktion_nimi> {lausekkeet}
```

Esimerkiksi jos haluamme avata Muistiolla (notepad.exe) teksti.txt tiedoston käskyllä *muistio* voimme tehdä sen näin.

```
Function muistio {notepad.exe c:\teksti.txt}
```

Funktiot voivat myös olla todella paljon monipuolisempiakin. Niitä voidaan käsitellä samalla tavalla kuin Cmdlet-komentojakin.

4.3.3 Komentosarjat

Komentosarjat tai skriptit ovat PowerShell koodia tekstitiedostossa, mikä nimetään yleensä .ps1 -päätteiseksi. Usein komentosarjat sisältävät useita komentoja. Komentosarjat ladataan ja jäsennetään jokainen kerta uudestaan, kun komentosarja ajetaan. Näin ollen niiden käynnistäminen on hieman funktioita hitaampaa, mutta ajonopeus funktion kanssa on sama sen jälkeen kuin komentosarja on ladattu muistiin. Muut ominaisuudet ovat samat funktioiden kanssa. (Payette & Siddaway 2018, 21)

4.3.4 Natiivit Windows-ohjelmat

Natiiveilla Windows-ohjelmilla tarkoitetaan tässä yhteydessä PowerShellin ulkopuolisia sovelluksia, esimerkiksi .exe -päätteisiä ajettavia sovelluksia. Jokaiselle ajettavalle sovellukselle luodaan oma prosessi, mikä aiheuttaa sen, että niiden käyttäminen PowerShellissä on hitain vaihtoehto. Myöskään PowerShellin sisäänrakennetut toiminnot eivät toimi näissä sovelluksissa. Jos ajetaan natiiveja Windows-sovelluksia, PowerShell odottaa, että natiivi prosessi on valmis tai sulkeutunut, ennen kuin se voi tehdä muuta. (Payette & Siddaway 2018, 22)

4.4 Aliakset

PowerShell tukee aliaksia, eli komennoille määriteltäviä vaihtoehtoisia kutsumanimiä tai lyhenteitä. Kaikki määritellyt aliakset näkyvät komennolla *Get-Alias*. Käyttäjä voi myös määritellä itse uusia aliaksia komennolla *Set-Alias*. *Set-Alias* komentoa käyttämällä määritelty alias pysyy voimassa vain käynnissä olevan istunnon ajan, eli jos käyttäjä sulkee PowerShellin katoavat tehdyt määrittelyt. Määrittelyt on mahdollista kuitenkin asettaa pysyviksi muokkaamalla PowerShell profiilitiedostoa.

Esimerkiksi DOS-maailmasta tuttu komento hakemiston vaihtamiseen *cd* eli change directory on PowerShellissä alias komennolle *Set-Location*. Tämä komento tekee saman asian ja onkin käyttäjäystävällistä, että se myös toimii vanhalla komennolla. Muita maininnan arvoisia aliaksia komennoille ovat: *del* (*Remove-Item*), *cp* ja *copy* (*Copy-Item*). Hakemiston sisällön listaamiseen käytetyt Linuxista tuttu komento *ls* ja DOS-puolelta tuttu *dir* ovat molemmat aliaksia komennolle *Get-Childitem*.

Uuden aliaksen luomiseen käytetään siis komentoa *Set-Alias*. Alla olevassa esimerkissä luodaan alias apua ajamaan cmdlet-komento *Get-Help*.

```
Set-Alias -Name apua -Value Get-Help
```

Parametri *name* määrittää nimen aliakselle ja parametri *value* on komento mihin alias osoittaa.

4.5 Objektit

PowerShell pitää tiedon jatkuvasti sen ominaisessa muodossaan objekteissa. Monissa muissa komentokehotteissa tieto muutetaan tekstimuotoon, mikä hankaloittaa tiedon käsittelyä. Koska PowerShell on rakennettu .Net-ohjelmistokomponenttikirjaston päälle tiedon synnynnäinen muoto on .NET-objektit. Jokaisella objektilla on ominaisuuksia ja metodeita. (Holmes 2013, 117)

Komento *Get-Member* näyttää objektin ominaisuudet.

```
PS C:\> Get-Service | Get-Member


   TypeName: System.ServiceProcess.ServiceController

Name                      MemberType    Definition
----                      ----------    ----------
Name                      AliasProperty Name = ServiceName
RequiredServices          AliasProperty RequiredServices = ServicesDependedOn
Disposed                  Event         System.EventHandler Disposed(System.Object, System.EventArgs)
Close                     Method        void Close()
Continue                  Method        void Continue()
CreateObjRef              Method        System.Runtime.Remoting.ObjRef CreateObjRef(type requestedType)
Dispose                   Method        void Dispose(), void IDisposable.Dispose()
Equals                    Method        bool Equals(System.Object obj)
ExecuteCommand            Method        void ExecuteCommand(int command)
GetHashCode               Method        int GetHashCode()
GetLifetimeService        Method        System.Object GetLifetimeService()
GetType                   Method        type GetType()
InitializeLifetimeService Method        System.Object InitializeLifetimeService()
Pause                     Method        void Pause()
Refresh                   Method        void Refresh()
Start                     Method        void Start(), void Start(string[] args)
Stop                      Method        void Stop()
WaitForStatus             Method        void WaitForStatus(System.ServiceProcess.ServiceControllerStatus desiredStat...
CanPauseAndContinue       Property      bool CanPauseAndContinue {get;}
CanShutdown               Property      bool CanShutdown {get;}
CanStop                   Property      bool CanStop {get;}
Container                 Property      System.ComponentModel.IContainer Container {get;}
DependentServices         Property      System.ServiceProcess.ServiceController[] DependentServices {get;}
DisplayName               Property      string DisplayName {get;set;}
MachineName               Property      string MachineName {get;set;}
ServiceHandle             Property      System.Runtime.InteropServices.SafeHandle ServiceHandle {get;}
ServiceName               Property      string ServiceName {get;set;}
ServicesDependedOn        Property      System.ServiceProcess.ServiceController[] ServicesDependedOn {get;}
ServiceType               Property      System.ServiceProcess.ServiceType ServiceType {get;}
Site                      Property      System.ComponentModel.ISite Site {get;set;}
StartType                 Property      System.ServiceProcess.ServiceStartMode StartType {get;}
Status                    Property      System.ServiceProcess.ServiceControllerStatus Status {get;}
ToString                  ScriptMethod  System.Object ToString();
```

Kuva 4: Komennon Get-Service ominaisuudet

Kuvassa 4 nähdään *Get-Service* komennon ominaisuudet ja metodit. Oletuksena komento *Get-Service* näyttää vain *Status*, *Name* ja *DisplayName* kohdat, mutta kaikki yllä oleva tieto on objektissa.
PowerShellissä on komentoja objektien käsittelyyn. Esimerkiksi komennolla *Sort-Object* voidaan tulosteet järjestää halutun ominaisuuden mukaan.

```
Get-Service | Sort-Object DisplayName
```

Esimerkissä haetaan kaikki palvelut ja sen jälkeen järjestetään ne *DisplayName*-ominaisuuden mukaan, eli tässä tapauksessa nimen mukaisessa järjestyksessä.

Toinen hyödyllinen komento objektien hallintaan on *Select-Object.* Komennossa annetaan ominaisuudet mitkä halutaan näyttää ja niistä syntyy uusi objekti. Esimerkissä halutaan nähdä palvelun nimi, tila ja palvelun tyyppi. Kuvassa 5 näkyy komennon tuloste.

```
Get-Service | Select-Object -Property DisplayName, Status,
ServiceType
```

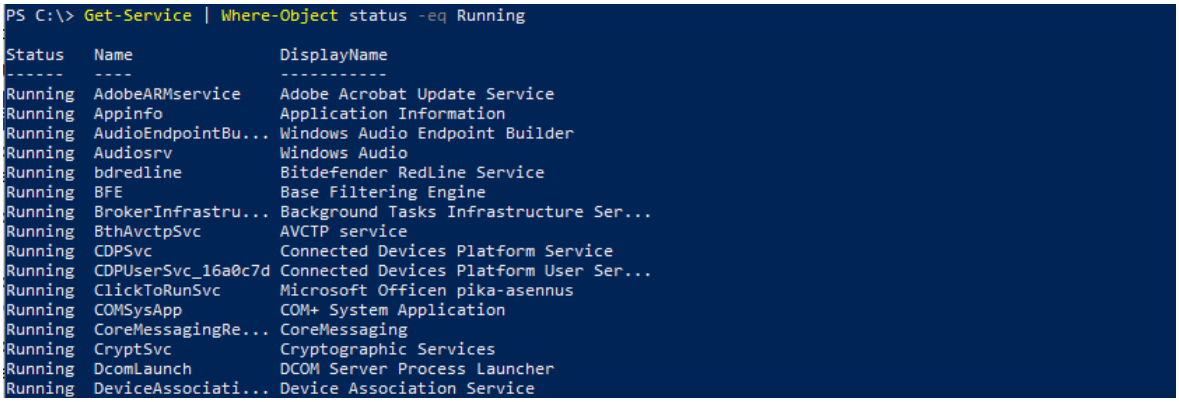
```
PS C:\> Get-Service | Select-Object -Property DisplayName,Status,ServiceType

DisplayName                                           Status                                ServiceType
-----------                                           ------                                -----------
Adobe Acrobat Update Service                          Running                             Win32OwnProcess
Adobe Flash Player Update Service                     Stopped                             Win32OwnProcess
AllJoyn Router Service                                Stopped                           Win32ShareProcess
Application Layer Gateway Service                     Stopped                             Win32OwnProcess
Application Identity                                  Stopped                           Win32ShareProcess
Application Information                               Running    Win32OwnProcess, Win32ShareProcess
Application Management                                Stopped                           Win32ShareProcess
App Readiness                                         Stopped    Win32OwnProcess, Win32ShareProcess
Microsoft App-V Client                                Stopped                             Win32OwnProcess
AppX Deployment Service (AppXSVC)                     Stopped    Win32OwnProcess, Win32ShareProcess
AssignedAccessManager Service                         Stopped                           Win32ShareProcess
Windows Audio Endpoint Builder                        Running    Win32OwnProcess, Win32ShareProcess
Windows Audio                                         Running                             Win32OwnProcess
ActiveX Installer (AxInstSV)                          Stopped                           Win32ShareProcess
GameDVR and Broadcast User Service_16a0c7d            Stopped                                         224
BitLocker Drive Encryption Service                    Stopped    Win32OwnProcess, Win32ShareProcess
Bitdefender RedLine Service                           Running                             Win32OwnProcess
Base Filtering Engine                                 Running                           Win32ShareProcess
Background Intelligent Transfer Service               Stopped                           Win32ShareProcess
Bluetooth User Support Service_16a0c7d                Stopped                                         224
Background Tasks Infrastructure Service               Running                           Win32ShareProcess
Bluetooth Audio Gateway Service                       Stopped                           Win32ShareProcess
AVCTP service                                         Running    Win32OwnProcess, Win32ShareProcess
Bluetooth Support Service                             Stopped                           Win32ShareProcess
Capability Access Manager Service                     Stopped    Win32OwnProcess, Win32ShareProcess
CaptureService_16a0c7d                                Stopped                                         224
```

Kuva 5: Select-Object toiminnassa

Viimeisenä esittelen komennon *Where-Object*. Kyseinen komento valitsee objektijoukosta haluttuja arvoja vastaavat objektit.

```
Get-Service | Where-Object status -eq Running
```

Esimerkin komento näyttää kaikki käynnissä olevat palvelut. Etsimisessä käytetään objektin ominaisuutta *status* ja vertailuoperaattoria *-eq* (equal) eli on yhtä suuri kuin.



Kuva 6: Where-Object tuloste

4.6 Komentojen putkitus

PowerShell mahdollistaa komentojen ajamisen sarjassa. Komennot eritellään niin kutsutulla putki-operaattorilla eli pystyviivalla (|). Jokainen komento putkessa saavat edellisestä komennosta syntyvän objektin, suorittavat jonkin operaation tälle objektille ja lähettävät sen taas eteenpäin seuraavalle komennolle. Objekti lähetetään eteenpäin heti kun on mahdollista eikä PowerShell jää odottamaan, että koko tulos olisi valmis.

```
Komento -Parametri1 Argumentti1 | Komento2 -Parametri1 Argumentti 1
```

Ylläolevassa esimerkissä suoritetaan ensin ensimmäinen komento minkä luoma objekti lähetetään komennolle 2. (Payette & Siddaway 2018, 35-37)

Käytännön esimerkki:

```
Get-Process -name Powershell | Stop-Process
```

Edeltävässä esimerkissä käytetään *Get-Process* komentoa etsimään prosessia nimeltä Powershell ja sen jälkeen käytetään komentoa *Stop-Process* lopettamaan kyseinen prosessi.

4.7 Etäkäyttö

PowerShell tarjoaa muutaman erilaisen tavan etäkäyttää järjestelmiä. Tarpeesta riippuen kannattaa valita sopiva.

Ensimmäinen tapa on niin kutsuttu klassinen etäkäyttö. Klassinen etäkäyttö käyttää DCOM ja RPC -protokollia luodakseen yhteyden. Etäkäytön mahdollistamiseksi palomuuriin tulee avata tarvittavia portteja, jotta yhteys voidaan luoda. Lukuisa joukko cmdlet-komentoja tukee klassista etäkäyttöä. Klassisessa etäkäytössä käyttäjältä kysytään käyttäjätunnuksia, jos käytössä olevalla tunnuksella ei ole oikeuksia tehtävään toimenpiteeseen.

Esimerkki klassisesta etäkäytöstä:

```
Get-Service -computername dc1 -name b*
```

Komento yrittää hakea koneelta nimeltä dc1 palveluita jotka alkavat b-kirjaimella. C*omputername*-parametrilla määritellään etäkäytettävän koneen nimi ja *Name*-parametrilla haettava palvelu.

Windows Remote Management (WinRM) on Windows palvelinkäyttöjärjestelmiin asennettava etäkäytön mahdollista ohjelmisto. WinRM otetaan käyttöön komennolla *Enable-PSRemoting*. Kyseinen komento ajaa *Set-WSManQuickConfig* cmdlet-komennon, mikä muun muassa käynnistää WinRM-palvelun, lisää sen automaattisesti käynnistyviin ohjelmiin, lisää kuuntelijan, mikä hyväksyy etäistuntoja ja lisää palomuuriin tarvittavat säännöt.

```
WinRM Quick Configuration
Running the Set-WSManQuickConfig command has significant security implications, as it enables remote management through
 the WinRM service on this computer.
This command:
 1. Checks whether the WinRM service is running. If the WinRM service is not running, the service is started.
 2. Sets the WinRM service startup type to automatic.
 3. Creates a listener to accept requests on any IP address. By default, the transport is HTTP.
 4. Enables a firewall exception for WS-Management traffic.
 5. Enables Kerberos and Negotiate service authentication.
Do you want to enable remote management through the WinRM service on this computer?
[Y] Yes  [N] No  [S] Suspend  [?] Help (default is "Y"):
```

Kuva 7: WinRM pikakonfiguraatio

Etäyhteyden toimivuutta voidaan kokeilla *Test-WSMan* komennolla.

```
PS C:\> Test-WSMan -ComputerName DC1


wsmid           : http://schemas.dmtf.org/wbem/wsman/identity/1/wsmanidentity.xsd
ProtocolVersion : http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd
ProductVendor   : Microsoft Corporation
ProductVersion  : OS: 0.0.0 SP: 0.0 Stack: 3.0
```

Kuva 8: Oikein muodostunut yhteys palauttaa kuvan mukaisen näkymän

Varsinainen etäsessio käynnistetään komennolla *Enter-PSSession*. Kyseinen komento avaa interaktiivisen etäkomentokehoteyhteyden hallittavalle koneelle. Istunto päättyy komennolla *exit*.

Kuva 9: Etäistunnon tunnistaa siitä, että etäkoneen nimi on rivin alussa hakasuluissa

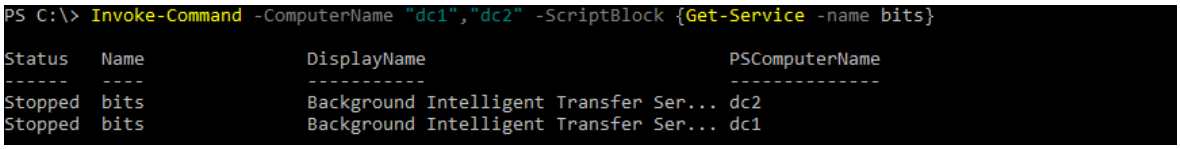
Yksittäisten komentojen lähettäminen hallittaville koneille on mahdollista komennolla *Invoke-Command.*

```
Invoke-Command -Computername dc1 -ScriptBlock {get-service bits |
stop-service}
```

Kohdekone määritellään C*omputername*-parametrilla ja lähetettävä komento tai komennot lisätään S*criptblock*-parametriin aaltosulkeiden { } sisälle.

Invoke-Command mahdollistaa myös useammalle kuin yhdelle koneelle komentojen lähettämisen. *ComputerName*-parametri ottaa vastaan useampia nimiä pilkulla eroteltuna. Esimerkissä on lähetetty komento koneille dc1 ja dc2 ja ajettu kohdekoneilla *Get-Service* kysely. (Wilson 2014, 615-633)

```
Invoke-Command -ComputerName "dc1","dc2" -ScriptBlock {Get-Service -
name bits}
```



Kuva 10: Kahdelle eri koneelle lähetetty etäkomento

4.8 Vertailuoperaattorit

PowerShellissä arvojen vertailu tapahtuu vertailuoperaattoreilla. Operaattorit jaetaan viiteen eri tyyppiin. Powershellissä ei käytetä ohjelmointikielistä tuttuja matemaattisia merkkejä vaan englannin kielestä tulevista sanoista muodostettuja lyhenteitä.

Yhtäsuuruusoperaattorit ovat seuraavat:

Operaattori	Kuvaus
-eq	On yhtä suuri kuin. (equals)
-ne	On eri suuri kuin. (not equals)
-gt	Suurempi kuin. (greater than)
-ge	Suurempi tai yhtä suuri kuin. (greater than or equal)

-lt	Pienempi kuin. (less than)
-le	Pienempi tai yhtä suuri kuin. (less than or equal)

Yhteensopivuusoperaattorit palauttavat *true* tai *false* arvon. Kaksi ensimmäistä eroavat kahdesta jälkimmäisestä siinä, että *match* ja *nomatch* käyttävät hakemiseen säännöllistä lauseketta (regular expression):

Operaattori	Kuvaus
-like	Palauttaa true-arvon, jos merkkijono löytyy haettavasta tekstistä. Voidaan käyttää jokerimerkkiä.
-notlike	Palauttaa true-arvon, jos merkkijonoa ei löydy haettavasta tekstistä. Voidaan käyttää jokerimerkkiä.
-match	Käyttää hakemiseen säännöllistä lauseketta. Palauttaa true-arvon, jos merkkijono sisältää haettavan tekstin.
-notmatch	Käyttää hakemiseen säännöllistä lauseketta. Palauttaa true-arvon, jos merkkijono ei sisällä haettavaa tekstiä.

Sisältää-operaattorit ovat samantapaisia kuin vertailuoperaattorit ja ne palauttavat arvon true tai false. Arvo palautetaan heti, jos vastaava arvo löytyy, loppuja ei tällöin käydä enää läpi. Operaattoreita ovat:

Operaattori	Kuvaus
-contains	Palauttaa arvon true, jos referenssiarvo löytyy joukosta.
-notcontains	Palauttaa arvon true, jos referenssiarvoa ei löydy joukosta.
-in	Palauttaa true, jos arvo löytyy referenssiarvoista. Tekee siis saman asian kuin contains, mutta toiste päin.
-notin	Palauttaa true, jos arvoa ei löydy referenssiarvoista. Tekee siis saman asian kuin contains, mutta toiste päin.

Muita vertailuoperaattoreita:
Replace-operaattorilla voidaan korvata kokonaan tai osittain arvoja. Esimerkissä korvataan t-kirjain s-kirjaimella.

```
”testi” -replace ”t”, ”s”
```

ja se palauttaa merkkijonon ”sessi”.

PowerShellistä löytyy myös tyyppien vertailuun operaattorit *-is* ja *-isnot*. Näitä käytetään objektien tyyppien vertailuun. (Microsoft 2017a)

4.9 Help-järjestelmä

PowerShell sisältää help-järjestelmän, mikä sisältää laajasti tietoa jokaiselle PowerShell-komennolle, PowerShellin toiminnasta sekä käsitteistä. Versiosta 3 eteenpäin help-tiedostot eivät ole oletuksena asennettuna valmiiksi, vaan ne tulee asentaa itse, näin käyttöön saadaan tuoreimmat versiot ohjeista. Cmdlet-komento *Update-Help* asentaa help-tiedostot ja päivittää ne myös ajan tasalle. (Payette & Siddaway 2018, 9)

Get-Help cmdlet-komento vaatii toimiakseen *Name*-parametrin. Kyseessä voi olla jonkin komennon koko nimi tai voidaan myös käyttää hakusanaa tai osaa siitä ja sen perässä jokerimerkkiä (*). Tyypillinen help-sivu sisältää lyhyen kuvauksen (synopsis), pidemmän kuvauksen (description), parametrien esittelyt, esimerkit (example), hakusanat, sekä aiheeseen liittyvät sivut (RelatedLinks).

```
Get-Help -name Get-Service
```

Yllä oleva komento tulostaa lyhyemmän yhteenvedon *Get-Service* komennon help-sivusta. Kun käytämme lisäksi parametria *-Detailed,* saadaan näkyviin parametrien selitykset sekä esimerkit, jos taas käytetään parametria *-Full,* saadaan esille koko ohjesivu kaikkine osioineen. Pelkät esimerkit saadaan esille käyttämällä parametria *-Examples*. Hyödyllinen parametri on myös *-ShowWindow*, mikä avaa ohjesivun ulkoiseen ikkunaan, missä on kätevä hakutoiminto. (Microsoft 2017b)

4.10 Tietoturva

Komentosarjakielet ovat pitkään olleet välineitä sähköpostipohjaisille haittaohjelmille Windows käyttöjärjestelmissä. PowerShelliä kehiteltäessä on otettu tämä huomioon alusta alkaen. Microsoft on tiedostanut PowerShellin houkuttelevuuden hyökkääjille jo ensimmäisen version julkaisusta lähtien, joten PowerShelliä onkin kehitetty tietoturva silmällä pitäen. Yhtenä huolenaiheena on ollut, se että PowerShellin lokitiedostot eivät ole olleet kovin kattavat. Joissain hyökkäystapauksissa lokitietojen tallennus ei ole ollut ollenkaan päällä. Tämä on mahdollistanut hyökkäysten tapahtumisen jälkiäkään jättämättä. Versiosta 5 eteenpäin PowerShellin lokitietojen tallennustyökaluja on parannettu tuntuvasti, mikä on parantanut hyökkäysten huomaamista ja tällöin myös niiden torjumista. Myös Windowsin suojaustyökalut tunnistavat koko ajan paremmin PowerShelliin liittyvää mah-

dollisesti tapahtuvaa epämääräistä toimintaa. Kuten kaikissa komentosarjakielissä ja komentokehotteista hyökkäyksiä ei täysin voida ehkäistä ja tietoturva-aukkoja paikataan jatkuvasti, mutta samalla uusia löydetään koko ajan. Tehokkaista hallintatyökaluista kuten PowerShell on kovasti hyötyä ylläpitäjille, mutta myös verkkorikollisillekin. Siksi onkin syytä huolehtia, että tietoturva on kunnossa kaikilta osin. (Newman 2017)

Yleinen luulo voi olla se, että komentokehotetta, tai komentosarjoja käytettäessä ohitettaisiin suojauksia, mitä Windowsin graafisessa käyttöliittymässä olisi, näin ei kuitenkaan ole. Windowsin suojausratkaisut suojaavat resursseja, ei miten niitä käytetään. PowerShell kuten muutkin Windowsin ohjelmat voivat tehdä asioita mihin niillä on oikeudet, siis samat oikeudet kuin käyttäjälläkin. Oletuksena PowerShellillä ei voida ajaa mitään komentosarjoja. Komentosarjojen ajamista haluavan käyttäjän pitää ottaa käyttöön kyseinen ominaisuus. Kyseessä on PowerShellin ExecutionPolicy käytäntö. Käytäntöä muutetaan komennolla *Set-ExecutionPolicy.* Valittavana on viisi eri käytäntöä:

1. Restricted: PowerShell käyttäytyy komentokehotteena, eikä komentosarjoja voida ajaa ollenkaan. Tämä on oletuksena päällä.
2. AllSigned: Vain digitaalisesti allekirjoitetut komentosarjat voidaan ajaa. PowerShell kysyy käyttäjältä tuntemattomista komentosarjoista, luotetaanko sen tekijään.
3. RemoteSigned: Useimmat komentosarjat ajetaan. Vain internetistä ladatut tarvitsevat digitaalisen allekirjoituksen ja tuntemattomista kysytään käyttäjältä luotetaanko niihin.
4. Unrestricted: Käytännössä ei vaadita digitaalista allekirjoitusta, mutta käyttäjälle annetaan varoitus internetistä ladatuista komentosarjoista.
5. Bypass: Komentosarjojen ajamisen suojaus on pois kokonaan käytöstä ja käyttäjälle jää kaikki vastuu.

Suojauskäytäntöä voidaan muuttaa antamalla komento ja jokin yllämainituista käytännöistä. Suositeltu käytäntö on *RemoteSigned*, näin käyttäjä voi ajaa omia koodipätkiä, mutta internetistä ladatuista vaaditaan digitaalinen allekirjoitus.

```
Set-ExecutionPolicy RemoteSigned
```

On kuitenkin hyvä muistaa, että allekirjoitettu koodinpätkä ei tarkoita välttämättä turvallista. Allekirjoitus mahdollistaa vain komentosarjan tekijän selvittämisen. Luonnollisesti koodin tekijän luotettavuutta ei tällä pystytä varmentamaan. (Holmes 2013, 515-519)

4.11 Tulostus tiedostoon

Välillä on tarve saada jonkin komennon tuloste talteen. PowerShell tarjoaa muutaman vaihtoehdon tähän tarpeeseen.

Out-File komento lähettää tulosteen tiedostoon. Sitä voidaan käyttää minkä tahansa cmdlet-komennon kanssa putken avulla. Jos *Out-File* -komennon syventäviä parametreja ei tarvita voidaan käyttää sen tilalle pelkkää operaattoria (>). Käytetään sitä esimerkissä:

```
Get-Service -name B* | Out-File -filepath c:\testi.txt
```

Samaa tarkoittaa myös:

```
Get-Service -name B* > c:\testi.txt
```

Yllä oleva komento listaa aluksi kaikki B-kirjaimella alkavat palvelut ja sen jälkeen *Out-File* -komennolla tulostetaan pakolliselle *Filepath*-parametrille annettuun sijaintiin txt-tiedostoon ASCII muodossa.

ConvertTo-Csv sekä *ConvertTo-Html* ovat hyödyllisiä cmdlet-komentoja, jos tarvitaan saada tulosteita ulos esimerkiksi taulukkolaskentasovellukseen tai selaimella käytettäväksi. Nimensä mukaisesti yllä mainitut komennot muuttavat tulosteen joko CSV (Comma-separated values) tai HTML (Hypertext Markup Language) muotoon. Komennot toimivat putken avulla ja oletuksena tulostavat tehdyn muunnoksen konsoliin, mutta *Out-File* -komentoa (tai sen lyhennettä, on suurempi kuin merkkiä >) hyödyntämällä saadaan tuloste tallennettua myös määriteltävään kohdetiedostoon.

```
Get-Service B* | ConvertTo-Html > c:\testi.html
```

Komennon tuloksena syntyy kuvan 11 mukainen html-tiedosto, mikä sisältää kaikki b-kirjaimella alkavat palvelut.

Name	RequiredServices	CanPauseAndContinue	CanShutdown	CanStop	DisplayName	DependentServices	MachineName	ServiceName	ServicesDependedOn	ServiceHandle
BcastDVRUserService_2b7949	System.ServiceProcess.ServiceController[]	False	False	False	GameDVR and Broadcast User Service_2b7949	System.ServiceProcess.ServiceController[]	.	BcastDVRUserService_2b7949	System.ServiceProcess.ServiceController[]	
BDESVC	System.ServiceProcess.ServiceController[]	False	False	False	BitLocker Drive Encryption Service	System.ServiceProcess.ServiceController[]	.	BDESVC	System.ServiceProcess.ServiceController[]	
bdredline	System.ServiceProcess.ServiceController[]	False	True	False	Bitdefender RedLine Service	System.ServiceProcess.ServiceController[]	.	bdredline	System.ServiceProcess.ServiceController[]	SafeServiceHand
BFE	System.ServiceProcess.ServiceController[]	False	False	True	Base Filtering Engine	System.ServiceProcess.ServiceController[]	.	BFE	System.ServiceProcess.ServiceController[]	
BITS	System.ServiceProcess.ServiceController[]	False	False	False	Background Intelligent Transfer Service	System.ServiceProcess.ServiceController[]	.	BITS	System.ServiceProcess.ServiceController[]	SafeServiceHand
BluetoothUserService_2b7949	System.ServiceProcess.ServiceController[]	False	False	False	Bluetooth User Support Service_2b7949	System.ServiceProcess.ServiceController[]	.	BluetoothUserService_2b7949	System.ServiceProcess.ServiceController[]	SafeServiceHand
BrokerInfrastructure	System.ServiceProcess.ServiceController[]	False	False	False	Background Tasks Infrastructure Service	System.ServiceProcess.ServiceController[]	.	BrokerInfrastructure	System.ServiceProcess.ServiceController[]	
BTAGService	System.ServiceProcess.ServiceController[]	False	False	False	Bluetooth Audio Gateway Service	System.ServiceProcess.ServiceController[]	.	BTAGService	System.ServiceProcess.ServiceController[]	SafeServiceHand
BthAvctpSvc	System.ServiceProcess.ServiceController[]	False	False	True	AVCTP service	System.ServiceProcess.ServiceController[]	.	BthAvctpSvc	System.ServiceProcess.ServiceController[]	SafeServiceHand
bthserv	System.ServiceProcess.ServiceController[]	False	False	False	Bluetooth Support Service	System.ServiceProcess.ServiceController[]	.	bthserv	System.ServiceProcess.ServiceController[]	SafeServiceHand

Kuva 11: PowerShellin luoma html-tiedosto

CSV-muodossa olevien tulosteiden tai tiedostojen luominen toimii tismalleen samalla tavalla. Kuvassa 12 on haettu palvelua nimeltä BITS ja muunnettu tuloste CSV-muotoon.

```
PS C:\> Get-Service BITS  | ConvertTo-Csv
#TYPE System.ServiceProcess.ServiceController
"Name","RequiredServices","CanPauseAndContinue","CanShutdown","CanStop","DisplayName","DependentServices","MachineName",
"ServiceName","ServicesDependedOn","ServiceHandle","Status","ServiceType","StartType","Site","Container"
"BITS","System.ServiceProcess.ServiceController[]","False","False","False","Background Intelligent Transfer Service","Sy
stem.ServiceProcess.ServiceController[]",".","BITS","System.ServiceProcess.ServiceController[]","SafeServiceHandle","Sto
pped","Win32ShareProcess","Manual",,
```

Kuva 12: CSV-muodossa oleva tuloste

4.12 Objektien ja tulosteiden hallintaa

PowerShellissä on monta valmista komentoa objektien ja tulosteiden hallintaan. Esittelen tässä kolme erilaista komentoa millä pääsee jo pitkälle. Kaikkia komentoja käytetään putken avulla pääkomentojen perään.

Format-List komento alustaa tulosteen listaksi missä jokainen ominaisuus on sijoitettuna omalle riville. Ominaisuuksia voidaan valita Property-parametrilla.

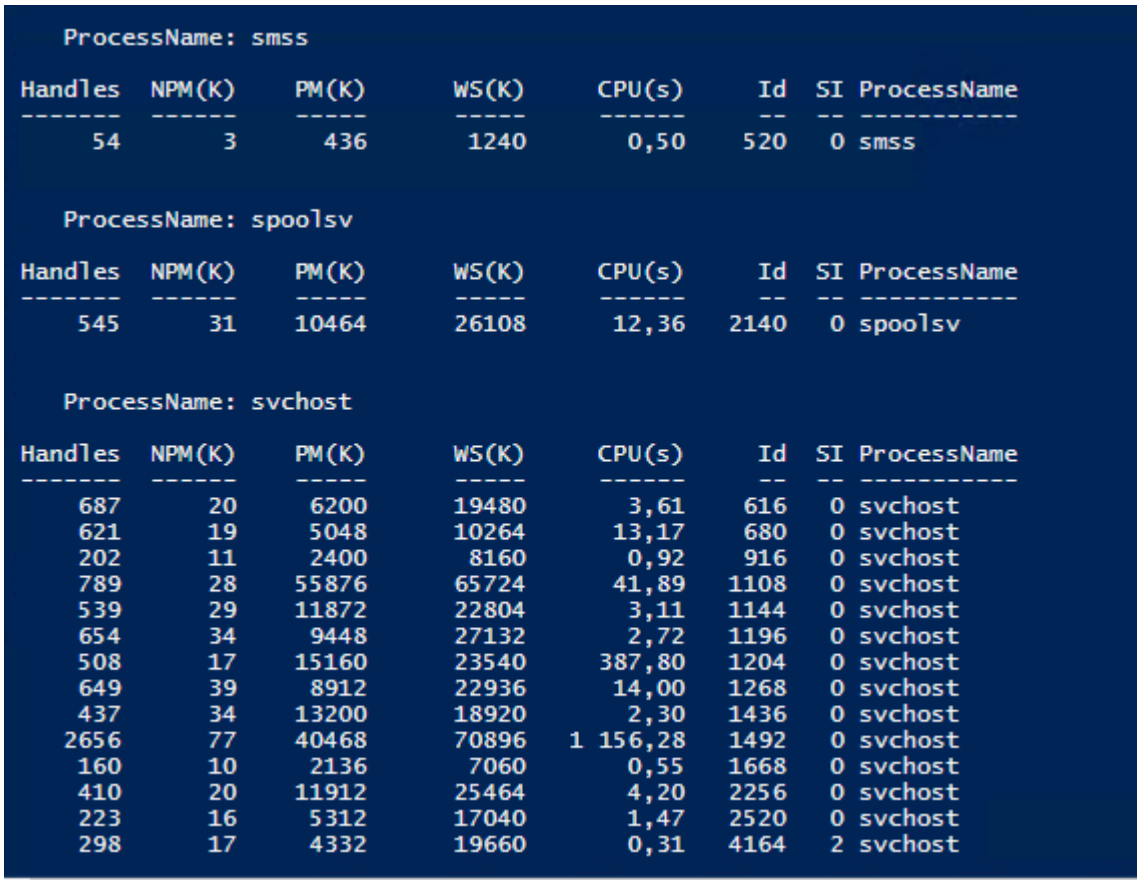
```
Get-Process powershell* | Format-List -Property Name,Id
```

```
PS C:\Windows\system32> Get-Process powershell* | Format-List -Property Name,Id


Name : powershell
Id   : 204

Name : powershell_ise
Id   : 5628

Name : powershell_ise
Id   : 5648
```

Kuva 13: Kaikki powershell-alkuiset prosessit listana omalla rivillänsä

Format-Table tekee tulosteesta taulukon valituille ominaisuuksille ja sijoittaa ominaisuuden omaan sarakkeeseen. Property-parametrilla määritellään näkyvät objektit. Muita hyödyllisiä parametreja *GroupBy*, mikä ryhmittelee rivit määritellyn ominaisuuden mukaan. *AutoSize* taas suurentaa automaattisesti sarakkeiden koon. Esimerkki ryhmittelee tulokset prosessin nimen mukaan ja lopputulos näkyy kuvassa 14.

```
Get-Process | Format-Table -GroupBy processname
```



```
   ProcessName: smss

Handles  NPM(K)    PM(K)      WS(K)     CPU(s)      Id  SI ProcessName
-------  ------    -----      -----     ------      --  -- -----------
     54       3      436       1240       0,50     520   0 smss


   ProcessName: spoolsv

Handles  NPM(K)    PM(K)      WS(K)     CPU(s)      Id  SI ProcessName
-------  ------    -----      -----     ------      --  -- -----------
    545      31    10464      26108      12,36    2140   0 spoolsv


   ProcessName: svchost

Handles  NPM(K)    PM(K)      WS(K)     CPU(s)      Id  SI ProcessName
-------  ------    -----      -----     ------      --  -- -----------
    687      20     6200      19480       3,61     616   0 svchost
    621      19     5048      10264      13,17     680   0 svchost
    202      11     2400       8160       0,92     916   0 svchost
    789      28    55876      65724      41,89    1108   0 svchost
    539      29    11872      22804       3,11    1144   0 svchost
    654      34     9448      27132       2,72    1196   0 svchost
    508      17    15160      23540     387,80    1204   0 svchost
    649      39     8912      22936      14,00    1268   0 svchost
    437      34    13200      18920       2,30    1436   0 svchost
   2656      77    40468      70896   1 156,28    1492   0 svchost
    160      10     2136       7060       0,55    1668   0 svchost
    410      20    11912      25464       4,20    2256   0 svchost
    223      16     5312      17040       1,47    2520   0 svchost
    298      17     4332      19660       0,31    4164   2 svchost
```

Kuva 14: Prosesseja ryhmitelty

Select-Object komennolla valitaan objekteja tai objektin ominaisuuksia. Ominaisuus valitaan Property-parametrilla. Esimerkissä valitaan komennosta *Get-Process* putken ja *Select-Object* komennon avulla vain *name* ja *id* ominaisuudet mistä objektit luodaan. Muut ominaisuudet tämän jälkeen eivät ole enää käytössä. Muita hyödyllisiä parametreja ovat: *Last* ja *First*, näillä valitaan rivien määrä alusta tai lopusta.

```
Get-Process | Select-Object -Property name,id
```

Sort-Object komennolla objekteja voidaan järjestellä annetuin säännöin. Parametrille *Property* annetaan ominaisuus, minkä mukaan järjestys luodaan. Esimerkissä prosessit järjestetään suoritinkäytön mukaan. Järjestystä voidaan muuttaa parametrilla Descending laskevaan järjestykseen.

```
Get-Process | Sort-Object -Property CPU
```

Lopuksi listataan prosessit suoritinkäytön mukaan nousevaan järjestykseen ja näytetään 10 suurinta tulosta ja valitaan ominaisuuksiksi nimi, id-numero sekä CPU.

```
Get-Process | Sort-Object -Property CPU -Descending | Select-Object
-Property Name,Id,CPU -First 10
```

```
PS C:\Windows\system32> Get-Process | Sort-Object -Property CPU -Descending | Select-Object -Property Name,Id,CPU -First 10

Name             Id        CPU
----             --        ---
MsMpEng        2624 1578,859375
svchost        1492   1154,9375
svchost        1204  385,296875
System            4  180,546875
powershell_ise 5628  151,453125
lsass           892  124,234375
WmiPrvSE       4080  120,828125
powershell_ise 5648   88,484375
WmiPrvSE       3576    70,03125
dfsrs          2320   49,640625
```

Kuva 15: 10 eniten suoritinaikaa käyttävää prosessia

5 Windows Server 2016 hallinta

5.1 Roolit ja ominaisuudet

Rooleja ja ominaisuuksia asennetaan ja poistetaan tehtävään suunnitelluilla Cmdlet-komennoilla. Asentamiseen tarvitaan komentoa: *Install-WindowsFeature.* Kyseinen komento tarvitsee toimiakseen parametriksi asennettavan toiminnon nimen ja sille voidaan myös antaa tarvittaessa muitakin parametreja.

Asennettavissa olevia rooleja pystytään hakemaan cmdlet-komennolla: *Get-WindowsFeature*. Komento tulee jokerimerkkiä, joten riittää, että tiedetään vain osa roolin tai ominaisuuden nimestä.

```
Get-WindowsFeature -name Web*
```

Esimerkki hakee kaikki web-alkuiset roolit ja ominaisuudet ja listaa ne ruudulle.

```
PS C:\Users\Administrator> Get-WindowsFeature web*

Display Name                                            Name                    Install State
------------                                            ----                    -------------
    [ ] Web Application Proxy                           Web-Application-Proxy       Available
[ ] Web Server (IIS)                                    Web-Server                  Available
    [ ] Web Server                                      Web-WebServer               Available
        [ ] Common HTTP Features                        Web-Common-Http             Available
            [ ] Default Document                        Web-Default-Doc             Available
            [ ] Directory Browsing                      Web-Dir-Browsing            Available
            [ ] HTTP Errors                             Web-Http-Errors             Available
            [ ] Static Content                          Web-Static-Content          Available
```

Kuva 16: Tuloste hakusanalle web*

Asennetaan esimerkkinä Windows IIS Web-Server kirjoittamalla seuraava cmdlet-komento:

```
Install-WindowsFeature Web-Server
```

Asennusohjelma ei kysy varmistusta, mutta se antaa loppuraportin asennuksen suoriutumisesta.

```
PS C:\Users\Administrator> Install-WindowsFeature web-server

Success Restart Needed Exit Code      Feature Result
------- -------------- ---------      --------------
True    No             Success        {Common HTTP Features, Default Document, D...
```

Kuva 17: Web-server ominaisuuden asennus

Roolien ja ominaisuuksien poistaminen tapahtuu komennolla *Remove-WindowsFeature.*

```
PS C:\Users\Administrator> Remove-WindowsFeature Web-Server

Success Restart Needed Exit Code      Feature Result
------- -------------- ---------      --------------
True    Yes            SuccessRest... {Common HTTP Features, Default Document, D...
WARNING: You must restart this server to finish the removal process.
```

Kuva 18: Web-server ominaisuuden poistaminen

Komennon suorittamisen jälkeen PowerShell antaa samanlaisen raportin kuin asentaessa. Tässä tapauksessa tulee palvelin myös käynnistää uudestaan poiston loppuun viemiseksi.

5.2 Ohjauskoneen määrittäminen ja siihen aktiivihakemisto-roolin asennus

Aktiivihakemiston käyttöönottoon ohjauspalvelimelle pitää asentaa oikea rooli ja konfiguroida se. Rooli, jonka haluamme asentaa on AD-Domain-Services.

```
Install-WindowsFeature -name AD-Domain-Services -
IncludeManagementTools
```

Esimerkki asentaa AD-Domain-Services (Active Directory Domain Services) roolin sekä parametrilla IncludeManagementTools asennetaan tarpeelliset Remote Server Administrator -työkalut.

Roolin asennuksen jälkeen se tulee konfiguroida jollain seuraavista komennoista:

1. Add-ADDSReadOnlyDomainControllerAccount
2. Install-ADDSDomain
3. Install-ADDSDomainController
4. Install-ADDSForest

Ensimmäinen komento on tarkoitettu palvelimille, joille on tarkoitus asentaa replikoiva ylikirjoitussuojattu ohjauspalvelinrooli. Komennolla on kolme pakollista parametria *DomainControllerAccountName*, *DomainName* sekä *SiteName*. Näillä määritetään muun muassa mitä palvelinta aletaan replikoimaan.

Toinen komento asentaa uuden aliosoitteen toimialueelle (child domain). Pakollisia parametrejä ovat *NewDomainName*, mikä määrittää uuden osoitteen nimen sekä parametri *ParentDomainName* vaaditaan määrittämään toimialueen nimi mihin aliosoite luodaan.

Kolmannella komennolla lisätään ohjauspalvelin jo luotuun aktiivihakemistoon. Pakollinen parametri komennolle on *DomainName,* millä määritetään mihin toimialueeseen palvelin lisätään.

Viimeinen komento asentaa ohjauspalvelimen ja määrittää sille uuden metsän. Pakollinen parametri on taas *DomainName*.

Esimerkissä haluamme asentaa ohjauspalvelimen ja luoda uuden metsän. Käytämme komentoa *Install-ADDSForest*, koska meillä ei ole vielä perustettu metsää ja haluamme sen perustaa. Yksinkertaisimmillaan komento toimii seuraavasti:

```
Install-ADDSForest -DomainName "testi.local" -InstallDNS
```

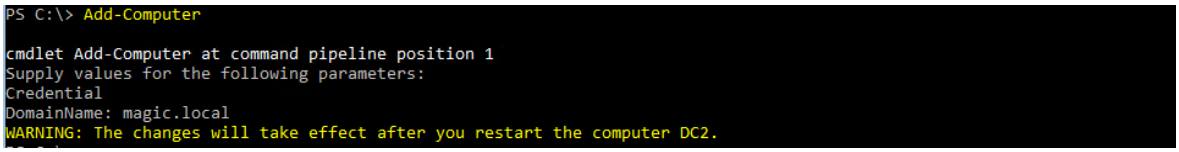
Komento luo metsän nimeltä “testi.local” ja asentaa myös DNS-nimipalvelun. Kaikkia asetuksia ei tarvitse syöttää, tällöin käytetään oletusarvoja. Komennon syöttämisen jälkeen PowerShell kysyy *SafeModeAdministrator*-salasanaa, millä voidaan muokata asetuksia jälkeenpäin. Kuva 19 esittää saman Windowsin Graafisella Server-managerilla tehtynä komentosarjana.

```
# Windows PowerShell script for AD DS Deployment
#

Import-Module ADDSDeployment
Install-ADDSForest `
-CreateDnsDelegation:$false `
-DatabasePath "C:\Windows\NTDS" `
-DomainMode "WinThreshold" `
-DomainName "testi.local" `
-DomainNetbiosName "TESTI" `
-ForestMode "WinThreshold" `
-InstallDns:$true `
-LogPath "C:\Windows\NTDS" `
-NoRebootOnCompletion:$false `
-SysvolPath "C:\Windows\SYSVOL" `
-Force:$true
```

Kuva 19: Server-managerin tekemä PowerShell komentosarja

5.3 Muita hyödyllisiä komentoja

Palvelin lisätään toimialueeseen kuvan 20 mukaisesti komennolla *Add-computer.*



Kuva 20: Komento kysyy käyttäjätietoja ja mihin toimialueeseen halutaan liittyä

Koneen nimen vaihtaminen onnistuu komennolla *Rename-Computer*. Palvelin käynnistetään uudestaan komennolla *Restart-Computer*.

Verkkoasetuksia tarkastellaan komennolla: *Get-NetIPConfiguration*. IP-osoitteen asetus tapahtuu *New-NetIPaddress* komennolla ja DNS-palvelin määritellään komennolla: *Set-DNSClientServerAddress.*

5.4 Tehtävien ajastaminen

Tehtävien ajastamiseen pätee PowerShellissä kolmivaiheinen malli. Aluksi luodaan tehtävä, sitten laukaisin ja lopuksi rekisteröidään. Esimerkissä halutaan sovellus notepad.exe käynnistymään joka päivä kello 12:00.

Aluksi luodaan muuttujat tehtävälle ja laukaisijalle, annetaan niille kuvaavat nimet: *action* sekä *trigger*.

Action-muuttujalle annetaan tehtäväksi suorittaa komento notepad.exe.

```
$action = New-ScheduledTaskAction -Execute "notepad.exe"
```

Sen jälkeen luodaan laukaisin (trigger). Laitetaan tehtävä toistumaan päivittäin klo 12:00. Muita mahdollisia parametreja ovat: käynnistyksen yhteydessä (AtStartup), kirjautumisen yhteydessä (AtLogOn), vain kerran (Once), viikottain (Weekly). At-parametria voidaan käyttää vain kalenteri- ja kellonaikapohjaisten aikojen kanssa.

```
$trigger = New-ScheduledTaskTrigger -Daily -At 12:00
```

Lopuksi hyödynnetään luotuja muuttujia ja rekisteröidään tehtävä ja annetaan sille nimi.

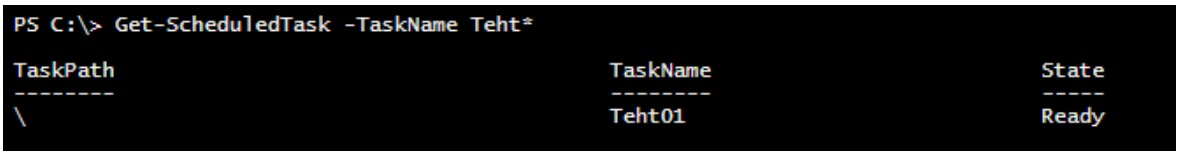
```
Register-ScheduledTask Teht01 -Action $action -Trigger $trigger
```

Ajastetun tehtävän poistaminen tapahtuu komennolla *Unregister-ScheduledTask.*

```
Unregister-ScheduledTask Teht01
```

Ajastettujen tehtävien selaaminen onnistuu komennolla *Get-ScheduledTask*. Tekemämme Teht01 löytyy esimerkiksi seuraavalla hakulauseella.

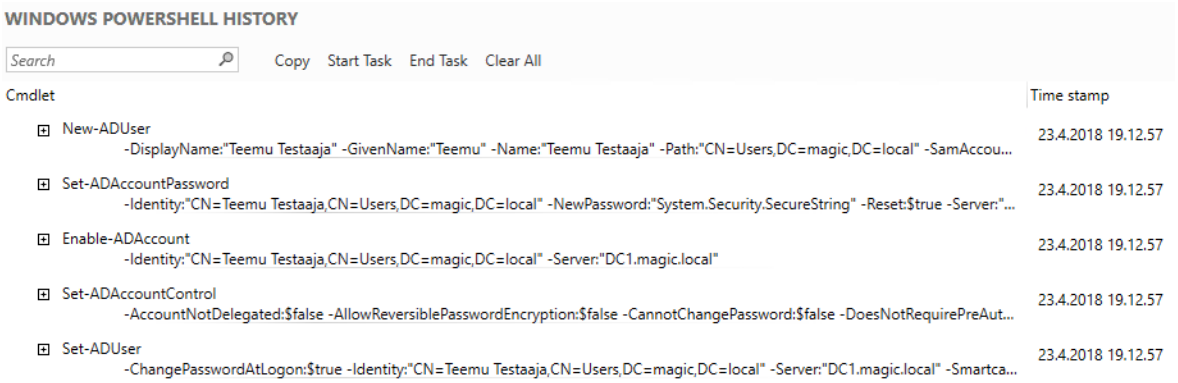
```
Get-ScheduledTask -TaskName Teht*
```



Kuva 21: Tehtävien hakeminen

6 Aktiivihakemiston hallinta

Kappaleessa esitellään yleisiä aktiivihakemiston ylläpitotehtäviä. Hallintaan voidaan käyttää graafista käyttöliittymää, Windows palvelinkäyttöjärjestelmien mukana toimitetaan Active Directory Administrative Center. Markkinoilla on myös ilmaisia ja maksullisia kolmannen osapuolen valmistamia sovelluksia aktiivihakemiston tehokkaaseen hallintaan. Yhteistä näissä kaikissa on se, että nämä sovellukset käyttävät konepellin alla PowerShell komentoja. Windows palvelinten mukana toimitettavassa ADAC-sovelluksessa jokaisen toiminnon jälkeen pystytään katsomaan mitä PowerShell komentoja käytettiin. Toiminto löytyy ohjelman alareunasta.



Kuva 22: Uuden käyttäjän luomisessa käytetyt PowerShell-komennot ja parametrit

6.1 Käyttäjähallinta

6.1.1 Käyttäjien lisääminen sekä poistaminen

Käyttäjät lisätään cmdlet-komennolla *New-ADUser*. Komennolla on yksi pakollinen parametri *name*, mikä määrittää nimen aktiivihakemiston objektille. Pelkällä *Name*-parametrilla luotu käyttäjä ei sisällä muita tietoja kuin objektin nimen.

```
New-ADUser -Name “Teemu Testaaja” -SamAccountName teemute -GivenName
Teemu -Surname Testaaja
```

Yllä oleva komento luo käyttäjän Teemu Testaaja, kirjautumistunnus määritellään parametrilla *SamAccountName*.

Jotta käyttäjätunnuksesta olisi jotain hyötyä pitää se vielä enabloida eli asettaa käyttökelpoiseksi. Tarve voi myös olla asettaa tunnukselle salasana ja lisätä se oikean organisaatioryhmään. Organisaatioyksikkö asetetaan *Path*-parametrilla.
Jos emme määritä mitään *Path*-parametrille käyttää palvelin oletusasetuksia. *Path*-parametri voisi kuitenkin olla seuraavanlainen:

```
-Path "OU=Users,dc=testi,dc=com"
```

Organisaatioyksiköksi tulee tällöin Users ja toimialueeksi testi.com.

Salasana asetetaan komennolla *Set-ADAccountPassword.* Pakollisen parametrin Identity arvoksi kelpaavat nimi, tunnusnumero (GUID), käyttöjärjestelmän generoima ObjectSid tai käyttäjätunnus (SamAccountName).

```
Set-ADAccountPassword -Identity teemute -NewPassword (ConvertTo-
SecureString -AsPlainText "Salasan@123" -Force)
```

Salasana muutetaan SecureString-muotoon siihen tarkoitetulla komennolla ennen kuin se tallennetaan. Muita hyödyllisiä parametrejä on *-ChangePasswordAtLogon,* mikä pakottaa käyttäjän vaihtamaan salasanan kirjautumisen yhteydessä, sekä tunnus voidaan enabloida suoraan lisäämällä parametri *-Enabled $true.*

Jos tunnusta ei ole luomisen yhteydessä enabloitu, se voidaan tehdä komennolla:

```
Enable-ADAccount -Identity "teemute"
```

Lopuksi vielä tehdään sama yhdellä komentojonolla. Luodaan käyttäjä, asetetaan sille käyttäjätunnus, nimi, salasana, salasananvaihtosääntö sekä enabloidaan tunnus.

```
New-ADUser -Name ”Testi Tunnus” -SamAccountName teemute -GivenName
Teemu -Surname Testaaja -AccountPassword (ConvertTo-SecureString -
AsPlainText "Salasan@123" -Force) -ChangePasswordAtLogon $True -
Enabled $True
```

Käyttäjätunnuksen poistaminen tapahtuu seuraavalla komennolla:

```
Remove-ADUser -Identity “käyttäjätunnus tai nimi”
```

6.1.2 Käyttäjien etsiminen

Get-ADUser on tehokas komento aktiivihakemiston käyttäjäobjektien etsimiseen. *Search-ADAccount* taas on hyödyllinen, jos halutaan etsiä käyttäjätunnuksia tiettyjen ehtojen mukaan kuten esimerkiksi lukittuneet tai vanhentuneet tunnukset.

Tietyn käyttäjän tietojen näyttäminen tapahtuu Identity-parametrilla:

```
Get-ADUser -Identity “tunnus”
```

Properties-parametrilla saadaan lisää tietoja, on mahdollista käyttää myös jokerimerkkiä, jolloin komento listaa kaikki mahdolliset kentät.

```
PS C:\Users\Administrator> Get-ADUser -Identity "teemute"


DistinguishedName : CN=Teemu Testaaja,CN=Users,DC=magic,DC=local
Enabled           : True
GivenName         : Teemu
Name              : Teemu Testaaja
ObjectClass       : user
ObjectGUID        : 80e553af-2b7c-4429-bc20-6637db7fa89b
SamAccountName    : teemute
SID               : S-1-5-21-1054666124-2548766112-3910391790-1108
Surname           : Testaaja
UserPrincipalName : teemute@magic.local
```

Kuva 23: Käyttäjän tiedot

Kaikki käyttäjät voidaan listata käyttämällä filter-parametria jokerimerkin kanssa.

```
Get-ADUser -Filter *
```

Hakujen suodattaminen tarjoaa monia ominaisuuksia. Esimerkiksi kaikki tunnukset, joiden nimi alkaa sanalla testi, voidaan hakea käyttämällä *like*-vertailijaa.

```
Get-ADUser -Filter 'Name -like "testi*"'
```

Search-ADAccount -komento tarjoaa tavan hakea tunnuksia tietyillä hakukriteereillä. Käytettävät hakuparametrit ovat:

AccountDisabled	Deaktivoidut tunnukset
AccountExpired	Vanhentuneet tunnukset
AccountExpiring	Vanhenevat tunnukset
AccountInactive	Epäaktiiviset tunnukset
LockedOut	Lukitut tunnukset
PasswordExpired	Tunnukset missä salasana vanhentunut
PasswordNeverExpires	Tunnukset missä salasana ei ikinä vanhene

Jos hakutulokset halutaan rajata pelkkiin käyttäjätunnuksiin tai tietokoneisiin, käytetään *UsersOnly* tai *ComputersOnly* -parametreja.

Aikaa määritellään joko *DateTime* tai *TimeSpan* -parametreilla. *DateTime* parametriin syötetään suoraan päivämäärä ja *TimeSpan* ottaa vastaan aika-arvon joko *TimeSpan*-objektina tai alemman esimerkin mukaisena arvona.

Etsitään esimerkissä tunnuksia, mitkä vanhenevat 30 päivän päästä hakuhetkestä. *AccountExpiring* käyttää *TimeSpan* parametria, millä määritellään aikajänne. Arvo voidaan syöttää joko käsin tai sitten voidaan käyttää komentoa *New-TimeSpan* mille annetaan *days*-parametri ja sille arvo 30 ja komento luo *TimeSpan*-objektin kuten kuvassa 24.

```
PS C:\> New-TimeSpan -days 30


Days              : 30
Hours             : 0
Minutes           : 0
Seconds           : 0
Milliseconds      : 0
Ticks             : 25920000000000
TotalDays         : 30
TotalHours        : 720
TotalMinutes      : 43200
TotalSeconds      : 2592000
TotalMilliseconds : 2592000000
```

Kuva 24: New-TimeSpan komennolla luodaan uusi timespan-objekti 30 päivälle

```
Search-ADAccount -AccountExpiring -TimeSpan (New-TimeSpan -days 30)
```

Timespan voidaan myös syöttää käsin. Ensimmäisenä tulee päivät, sitten tunnit ja minuutit sekä lopuksi sekunnit.

```
Search-ADAccount -AccountExpiring -TimeSpan 30.00:00:00
```

Toisessa esimerkissä etsitään deaktivoituja tunnuksia. Niitä voidaan hakea kahdella tavalla ja lopputulos on sama kuten kuvissa nähdään. Ensimmäisenä käytetään *Get-ADUser* -komentoa. Komennon kanssa käytetään *Filter*-parametria millä etsitään Enabled ominaisuus, minkä arvo on yhtä suuri kuin ”False”. Siistitään myös tulostetta *Format-List* komennolla näyttämään vain käyttäjänimet.

```
Get-ADUser -Filter {Enabled -eq "False"} | Format-List Name
```

```
PS C:\> Get-ADUser -Filter {Enabled -eq "False"}  | Format-List Name

Name : Guest

Name : DefaultAccount

Name : krbtgt

Name : teemu
```

Kuva 25: Ensimmäinen tapa tehdä haku

Jälkimäisessä käytetään *Search-ADAccount* -komentoa.

```
Search-ADAccount -AccountDisabled | Format-List Name
```

```
PS C:\> Search-ADAccount -AccountDisabled | Format-List Name

Name : krbtgt

Name : teemu

Name : Guest

Name : DefaultAccount
```

Kuva 26: Toinen tapa tehdä sama haku

Komennot tekivät saman asian, tulostusjärjestys on vain eri.

6.1.3 Käyttäjän muokkaus Set-ADUser komennolla

Set-ADUser cmdlet-komento on tarkoitettu aktiivihakemiston käyttäjien tunnusten ominaisuuksien muokkaamiseen. Käytössä on yli 60 eri parametria AD-tunnuksen muokkaamiseen. Muokattava käyttäjätunnus määritellään *Identity*-parametrilla käyttämällä käyttäjätunnuksen nimeä, GUID-tunnusta, SID-tunnusta tai SAM-tunnusta eli käyttäjän kirjautumiseen käytettävää lyhyttä käyttäjätunnusta. Identity-parametri on positionaalinen, joten sitä ei ole pakko kirjoittaa. Muita kuin komennon tarjoamia arvoja, mitä aktiivihakemistosta löytyy kymmeniä, muutetaan *add*, *remove*, *replace* ja *clear* parametreilla ja näissä parametreissa arvot syötetään hajautustaulun sisälle, mistä alempana on esimerkki.

Kaikki komennon käytettävissä olevat parametrit voidaan katsoa *Get-Help* -komennolla:

```
Get-Help Set-ADUser -Parameter *
```

Tunnuksen aktivointi:

```
Set-ADUser -Identity “kayttajatunnus” -Enabled $False
```

Tunnuksen kytkeminen pois päältä:

```
Set-ADUser -Identity kayttajatunnus -Enabled $True
```

Tunnuksen attribuuttien muuttaminen:

Ensimmäisessä esimerkissä korvataan käyttäjätunnuksen sähköpostiosoite, puhelinnumero, sekä kaupunki käyttämällä *Set-ADUser* komennon tarjoamia parametreja.

```
Set-ADUser -Identity kayttajatunnus -EmailAddress "testi@tunnus.fi"
-MobilePhone "050123123" -City "Turku"
```

Toisessa esimerkissä muutetaan arvoja mitä valmiilla parametreilla ei voida muuttaa. Käytetään *replace*-parametria mihin arvot syötetään hajautustauluun puolipilkulla eroteltuina. Esimerkissä asetetaan käyttäjälle huoneen numero ja kuva.

```
Set-ADUser -Identity kayttajatunnus -Add @{roomnumber="2";
photo="c:\kuva.gif"}
```

6.1.4 Tunnuksen lukituksen avaaminen

Jos tunnus on mennyt lukkoon esimerkiksi liian monen väärin yritetyn kirjautumisen takia, saadaan se auki seuraavalla komennolla:

```
Unlock-ADAccount -Indentity kayttajatunnus
```

6.1.5 Tunnuksen osallisuus eri ryhmiin

Aktiivihakemiston tunnusten osallisuus eri ryhmiin voidaan selvittää käyttämällä *Get-ADUser* komentoa. Koska ryhmiin kuuluminen ei näy oletuksena Get-ADUser hakutuloksessa, Properties-parametrilla valitaan ominaisuus *MemberOf*, mikä aktiivihakemistossa kertoo osallisuudet ryhmiin. Tulosteen selkeyttämiseksi käytetään myös *Select-Object* -komennon *ExpandProperty* parametria valitsemaan vain *MemberOf*-rivit.

```
Get-ADUser kayttajatunnus -Properties MemberOf | Select-Object -
ExpandProperty MemberOf
```

6.1.6 Käyttäjän salasanan nollaus

Salasanojen hallinta tapahtuu komennolla *Set-ADAccountPassword*. Jos käyttäjä on unohtanut salasanansa, voidaan se nollata ja asettaa uusi kyseisellä komennolla. Jos vanhaa salasanaa ei ole tiedossa käytetään *Reset*-parametria *NewPassword*-parametrin kanssa. Ilman *Reset*-parametria tulee käyttää *OldPassword*-parametria määrittämään käyttäjän vanha salasana. *Identity*-parametri täsmentää mihin tunnukseen muutos tehdään. Salasana muutetaan SecureString muotoon edempänä tutuksi tulleella tavalla. *Force*-parametria käytetään, koska halutaan yli kirjoittaa vanha salasana ilman varoituksia.

```
Set-ADAccountPassword -Identity kayttajatunnus -Reset -NewPassword
(ConvertTo-SecureString -AsPlainText "Salasan@123" -Force)
```

6.2 Ryhmien hallinta

6.2.1 Ryhmien luominen ja poistaminen

Uusi AD-ryhmä luodaan komennolla *New-ADGroup*. Komennolla on kaksi pakollista parametria *GroupScope* millä määritellään ryhmän näkyvyys sekä *Name* millä määritellään

ryhmä objektin nimi. Esimerkissä luodaan ryhmä nimeltä Testaus. Parametri *SamAccountName* määrittää ryhmälle käyttäjänimen. *GroupCategory* parametrilla kerrotaan ryhmän tyyppi, tässä tapauksessa *security*. *GroupScope*-parametrilla määritetään näkyvyys Globaliksi. *Path*-parametrilla määritellään organisaatioyksikkö, jos tähän ei määritetä mitään, käyttää PowerShell oletusarvoa. Tässä esimerkissä käytetään testiympäristön toimialuetta (testi.local) ja organisaatioyksikköä Suomi. Lopuksi *Description*-parametrilla annetaan ryhmälle kuvaus.

```
New-ADGroup -Name "Testaus" -SamAccountName Testaus -GroupCategory
Security -GroupScope Global -DisplayName "Testaajat" -Path
"ou=Suomi,dc=testi,dc=local" -Description "Kaikki testausryhmän
käyttäjät"
```

Ryhmä poistetaan komennolla *Remove-ADGroup*. Pakollinen parametri komennolle on Identity millä määritetään mikä ryhmä poistetaan. Identity voi olla esimerkiksi ryhmän nimi tai *SamAccountName*. Äsken luodun ryhmän poistaminen tapahtuu seuraavanlaisesti.

```
Remove-ADGroup -Identity Testaus
```

6.2.2 Käyttäjien lisäys ja poistaminen ryhmästä

Käyttäjiä lisätään ryhmiin komennolla *Add-ADGroupMember.* Komennolla on kaksi pakollista parametria: *Identity*, millä määritellään ryhmä sekä *Members* millä määritetään käyttäjät, jotka ryhmään lisätään. Esimerkissä lisätään ryhmään Testaus kolme uutta jäsentä.

```
Add-ADGroupMember -Identity Testaus -Members
teemu,testitunnus,testitunnus2
```

Käyttäjiä ryhmistä poistetaan samalla kaavalla käyttäen komentoa *Remove-ADGroupMember*. Esimerkissä poistetaan äsken lisätty tunnus teemu ryhmästä Testaus.

```
Remove-ADGroupMember -Identity Testaus -Members teemu
```

6.2.3 Näytä ryhmän jäsenet

Ryhmien jäseniä voidaan tarkastella komennolla:

```
Get-ADGroupMember -Identity Testaus
```

Komento listaa kuvan 27 mukaisesti ryhmän kaikki jäsenet.

```
PS C:\> Get-ADGroupMember -Identity Testaus


distinguishedName : CN=Testaaja T,OU=Suomi,DC=magic,DC=local
name              : Testaaja T
objectClass       : user
objectGUID        : 4a27ba72-8d75-400a-b288-4dc675c6e54f
SamAccountName    : testitunnus
SID               : S-1-5-21-1054666124-2548766112-3910391790-1130

distinguishedName : CN=testaaja kakkonen,OU=Suomi,DC=magic,DC=local
name              : testaaja kakkonen
objectClass       : user
objectGUID        : 0a4baac4-82ba-4272-be04-d92a1f61a3e3
SamAccountName    : testitunnus2
SID               : S-1-5-21-1054666124-2548766112-3910391790-1131
```

Kuva 27: Ryhmän jäsenet

6.2.4 Ryhmän ominaisuuksien muokkaaminen

Set-ADGroup komennolla muokataan ryhmän ominaisuuksia. Identity-parametrilla määritetään muokattava ryhmä. Muokataan esimerkissä ryhmän kuvausta.

```
Set-ADGroup -Identity Testaus -Description "Lähes kaikki testiryhmän jäsenet"
```

6.2.5 Ryhmän tietojen näyttäminen

Get-ADGroup näyttää ryhmän tiedot. Oletuksena tulosteessa ei näy ryhmän kuvausta. Kuvauksen saa näkyviin *Properties*-parametrilla.

```
Get-ADGroup -Identity Testaus -Properties description
```

```
PS C:\> Get-ADGroup -Identity Testaus -Properties description


Description       : Lähes kaikki testiryhmän jäsenet
DistinguishedName : CN=Testaus,OU=Suomi,DC=magic,DC=local
GroupCategory     : Security
GroupScope        : Global
Name              : Testaus
ObjectClass       : group
ObjectGUID        : a6b54e29-cfcd-47ea-976f-863536af9fb3
SamAccountName    : Testaus
SID               : S-1-5-21-1054666124-2548766112-3910391790-1129
```

Kuva 28: Ryhmän tiedot

6.3 Tietokoneiden hallinta

Jos tietokone liitetään toimialueeseen järjestelmä myös luo siitä uuden tietokone-objektin. On myös mahdollista lisätä tietokoneita käsin hyödyntäen PowerShellin *New-ADComputer* komentoa. Tämä voi olla tarpeellista jossain erikoisemmissa tilanteissa esimerkiksi jos kone pitää liittää toimialueeseen ilman verkkoyhteyttä (offline domain join). Komento toimii

samalla tavalla kuin uusien käyttäjien lisääminen komennolla *New-ADUser*. Komennossa ainoa pakollinen parametri on *Name*, jos muita ei käytetä täydentyvät ne oletusarvoin.

Esimerkissä luodaan kone nimeltä desktop1, myös koneen *SamAccountName* on desktop1. Kone sijoitetaan testiympäristöön Tietokoneet nimiseen organisaatioyksikköön.

```
New-ADComputer -Name desktop1 -SamAccountName desktop1 -Path
"ou=Tietokoneet,dc=testi,dc=local"
```

Tietokone-objekti poistetaan komennolla *Remove-ADComputer Identity*-parametria käyttäen.

```
Remove-ADComputer -Identity desktop1
```

Tietokoneen tietoja tarkastellaan komennolla: *Get-ADComputer*. Komennolle syötetään parametreiksi joko *Identity*, *LDAPFilter* tai *Filter*. Identity parameterille syötetään tietokoneen nimi. Filter-parametrit tarjoavat tavan hakea koneita erilaisin hakuehdoin. Ensimmäisessä esimerkissä haetaan tiettyä konetta ja halutaan nähdä koneen kaikki ominaisuudet, siksi käytetään *Properties*-parametria jokerimerkillä. PowerShell tulostaa usean kymmentä riviä tietoja koneesta.

```
Get-ADComputer -Identity desktop1 -Properties *
```

Toisessa esimerkissä käytetään *Filter*-parametria ja jokerimerkkiä, mikä listaa kaikki toimialueen koneet.

```
Get-ADComputer -Filter *
```

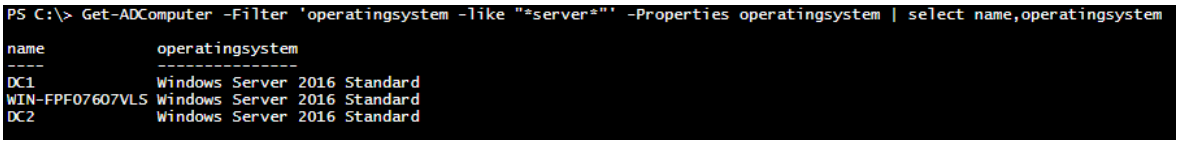
Kolmannessa esimerkissä suodatetaan koneita niiden nimen mukaan. Jos koneen nimestä löytyy merkkijono ”win”, tällöin se näytetään listauksessa.

```
Get-ADComputer -Filter 'name -like "*win*"'
```

Viimeisessä esimerkissä haetaan kaikkia koneita, joiden käyttöjärjestelmän nimessä on merkkijono ”server”. Tämä on nopea keino saada kaikkien toimialueen palvelinkoneiden tiedot.

```
Get-ADComputer -Filter 'operatingsystem -like "*server*"' -
Properties operatingsystem
```

Suodattamiseen voidaan käyttää siis kaikkia tietokone-objektin ominaisuuksia ja vertailuun PowerShellin tukemia operaattoreita. Hakutuloksia kannattaa hieman siistiä käyttämällä esimerkiksi *Select-Object* -komentoa kuten kuvassa 29 on tehty.



Kuva 29: Siistitty listaus toimialueen palvelinkoneista

6.4 Organisaatioyksiköt

Organisaatioyksiköitä selataan komennolla: *Get-ADOrganizationalUnit.* Komento toimii samalla tavalla kuin edeltävän kappaleen *Get-ADComputer*. Eli voidaan käyttää joko *Identity*-parametriä, jos tiedetään organisaatioyksikön nimi tai *Filter*-parametria organisaatioyksiköiden etsimiseen jollain määriteltävillä ehdoilla.

Esimerkissä listataan kaikki organisaatioyksiköt antamalla parametrille *Filter* hakuehdoksi jokerimerkki.

```
Get-ADOrganizationalUnit -Filter *
```

Toisessa esimerkissä haetaan tiedetyn organisaatioyksikön kaikki tiedot. *Identity*-parametrille annetaan organisaatioyksikön tarkka sijainti ja P*roperties*-parametrille jokerimerkki jolloin tulostetaan kaikki ominaisuudet.

```
Get-ADOrganizationalUnit -Identity "OU=Suomi,DC=magic,DC=local" -
Properties *
```

Lopuksi etsitään organisaatioyksiköitä, joille on asetettu maaksi Suomi. Haku palauttaa kuvan 30 mukaisen tulosteen.

```
Get-ADOrganizationalUnit -Filter 'Country -like "FI"'
```

```
PS C:\>
Get-ADOrganizationalUnit -Filter 'Country -like "FI"'


City                    :
Country                 : FI
DistinguishedName       : OU=Suomi,DC=magic,DC=local
LinkedGroupPolicyObjects : {}
ManagedBy               :
Name                    : Suomi
ObjectClass             : organizationalUnit
ObjectGUID              : 96e87b32-4279-4886-856c-53229f846d2f
PostalCode              :
State                   :
StreetAddress           :
```

Kuva 30: Organisaatioyksiköiden suodatusta

Uusi organisaatioyksikkö luodaan komennolla *New-ADOrganizationalUnit*. Pakollinen parametri on *Name*, eli organisaatioyksikön nimi. Jos halutaan määrittää organisaatioyksikölle tarkka sijainti, se määritellään *Path*-parametrilla, muuten käytetään oletussijaintia.

Organisaatioyksiköt poistetaan komennolla *Remove-ADOrganizationalUnit.* Komennolle annetaan organisaatioyksikön tarkka sijainti *Identity*-parametrilla.

7 Uusien tunnusten luominen: PowerShell vs. AD-Manager

PowerShellin vahvuus tulee ilmi tehtävissä mitä pitää tehdä usein. Esimerkiksi uusien käyttäjien luominen toimialueelle on ylläpitotehtävä mitä joutuu tekemään yrityksissä aina uusien työntekijöiden saapuessa. Tällöin kannattaakin miettiä olisiko järkevämpää käyttää hieman aikaa siihen, että luodaan komentosarja, millä uusien käyttäjien lisäys on nopeampaa kuin käsin. Skriptin kirjoittamisessa voi mennä aikaa enemmän, kuin yhden uuden käyttäjän lisäämisessä käsin mutta säästetty aika jatkossa voi olla merkittävä.

Tämän kappaleen tarkoitus on esitellä yksinkertainen komentosarja millä toimialueelle voidaan luoda uusia käyttäjiä. Komentosarja toimii myös hyvänä esimerkkinä kuinka yksinkertainen ja samaan aikaan tehokas PowerShellin komentosarja voi olla. Esittelen myös miten komentosarjan toiminta toistetaan graafisessa käyttöliittymässä. Kappaleen tarkoituksena, ei ole väittää, että kaikki toiminnot olisivat paras tehdä PowerShellillä, vaan tarkoituksena on osoittaa, että toistuvien tehtävien tekeminen PowerShellillä voi säästää paljonkin aikaa.

Esimerkkikomentosarjassa luodaan uusi AD-käyttäjä, määritetään sille salasana ja siirretään se haluttuun organisaatioyksikköön ja jonkin ryhmän jäseneksi. Sama asia tehdään PowerShell skriptinä sekä graafisesti AD-managerilla.

Read-Host komento *Promt*-parametrilla kysyy käyttäjältä arvon, mikä sijoitetaan muuttujiin. Olen nimennyt kaikki muuttujat kuvaavasti. Muuten komentosarjassa käytetään aiemmin tutuiksi tulleita komentoja. Perustelut miksi uuden käyttäjän tiedot tallennetaan väliaikaisesti uusikayttaja.txt tiedostoon on seuraava: usein uudelle käyttäjälle tulee ilmoittaa tehdyt tunnukset kirjautumista varten esimerkiksi kirjallisesti tai sähköpostilla. Tällöin ne ovat tallessa tiedostossa niin kauan kuin tehdään uusi käyttäjä ja vanhat tiedot yli kirjoitetaan. Tämä ei ole tietoturvariski sinänsä, koska käyttäjän tulee vaihtaa salasanansa ensimmäisen kirjautumisen yhteydessä. Käyttäjätunnus muodostetaan PowerShelliin sisäänrakennetulla merkkijonojen käsittelyyn tehdyllä *SubString*-metodilla, mikä valitsee etunimestä ja sukunimestä 3 kirjainta ja muodostaa niistä lyhyen käyttäjätunnuksen.

Komentosarjaa voisi jatkokehittää paljonkin, esimerkiksi lähettämään tunnukset johonkin sähköposti osoitteeseen tai tulostumaan suoraan paperille. Myös useampaan ryhmään lisääminen voisi olla mahdollista hieman erilaisella lähestymistavalla. Lisäksi tarkistukset jo käytössä oleville nimille ja käyttäjätunnuksille olisivat hyödyllisiä, nyt komentosarja yrittää tehdä tunnukset ja jos ne löytyvät jo päättyy komentosarjan suorittaminen virhetilaan.

Komentosarjaan voi asettaa parametreina muuttujille arvot. Parametrit määritellään *Param*-lohkossa esimerkin tavoin. Mahdollista olisi myös määritellä joitain parametreja pakolliseksi, mutta tässä tapauksessa niitä ei tarvita. Komentosarjaa voi käyttää myös ilman parametreja, tällöin komentosarja kysyy käyttäjältä arvot yksitellen. Olisi myös mahdollista jättää *Read-Host* -komennot pois, mutta tällöin arvojen kysyminen käyttäjälle tapahtuisi vain, jos arvo olisi määritelty pakolliseksi.

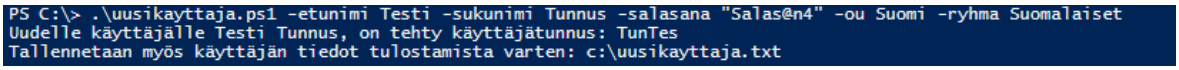
```
Param(
  [string]$etunimi = (Read-Host -Prompt "Kirjoita käyttäjän
etunimi"),
  [string]$sukunimi = (Read-Host -Prompt "Kirjoita käyttäjän
sukunimi"),
  [string]$salasana = (Read-Host -Prompt "Kirjoita käyttäjälle
salasana"),
  [string]$ou = (Read-Host "Minne ou:hun käyttäjä lisätään"),
  [string]$ryhma = (Read-Host "Minne ryhmaan käyttäjä lisätään")
)
$tiedostosijainti = "c:\uusikayttaja.txt"
$kayttajanimi = $sukunimi.Substring(0,3) + $etunimi.Substring(0,3)

New-ADUser -Name "$etunimi $sukunimi" -SamAccountName $kayttajanimi
-GivenName $etunimi -Surname $sukunimi -AccountPassword (ConvertTo-
SecureString -AsPlainText $salasana -Force) -ChangePasswordAtLogon
$True -Enabled $True -Path "OU=$ou,dc=magic,dc=local"
Add-ADGroupMember -Identity "$ryhma" -Members $kayttajanimi
```

```
Write-Host "Uudelle käyttäjälle $etunimi $sukunimi, on tehty
käyttäjätunnus: $kayttajanimi"
Write-Host "Tallennetaan myös käyttäjän tiedot tulostamista varten:
$tiedostosijainti"
$etunimi,$sukunimi,$kayttajanimi,$salasana | Out-File -FilePath
$tiedostosijainti
```

Komentosarja arvot parametreina syötettynä toimii seuraavalla logiikalla:

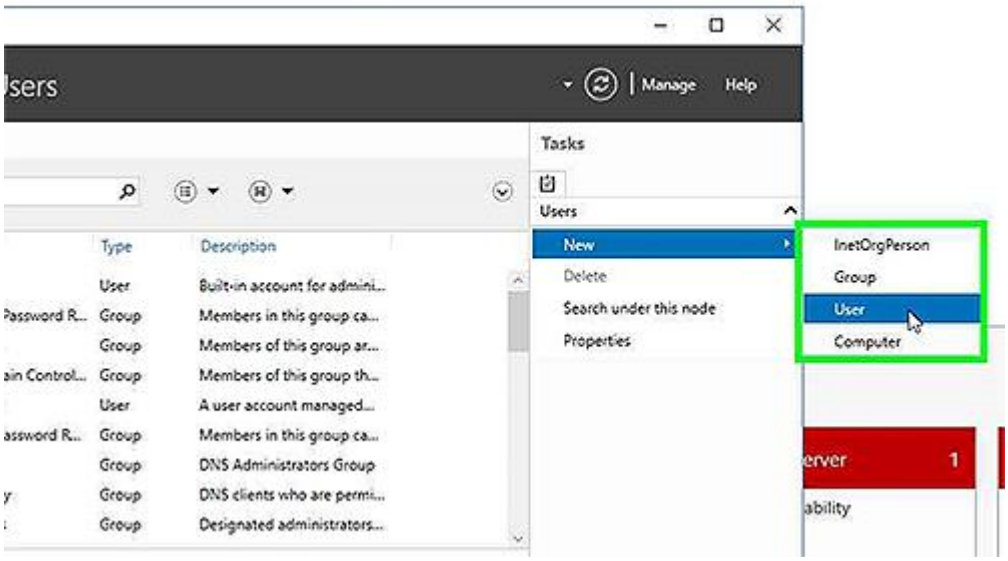
```
.\uusikayttaja.ps1 -etunimi Testi -sukunimi Tunnus -salasana
"Salas@n4" -ou Suomi -ryhma Suomalaiset
```

Komentosarja ajetaan kirjoittamalla tiedoston eteen piste sekä kenoviiva. Komento luo käyttäjän Testi Tunnus, määrittää sille salasanan sekä lisää käyttäjän organisaatioyksikköön Suomi ja ryhmään Suomalaiset. Kuten kuvasta 31 näkyy komentosarja tulostaa onnistuneesta käyttäjätunnuksen luomisesta uuden käyttäjätunnuksen ja tiedoston sijainnin mistä loput tiedot löytyvät.

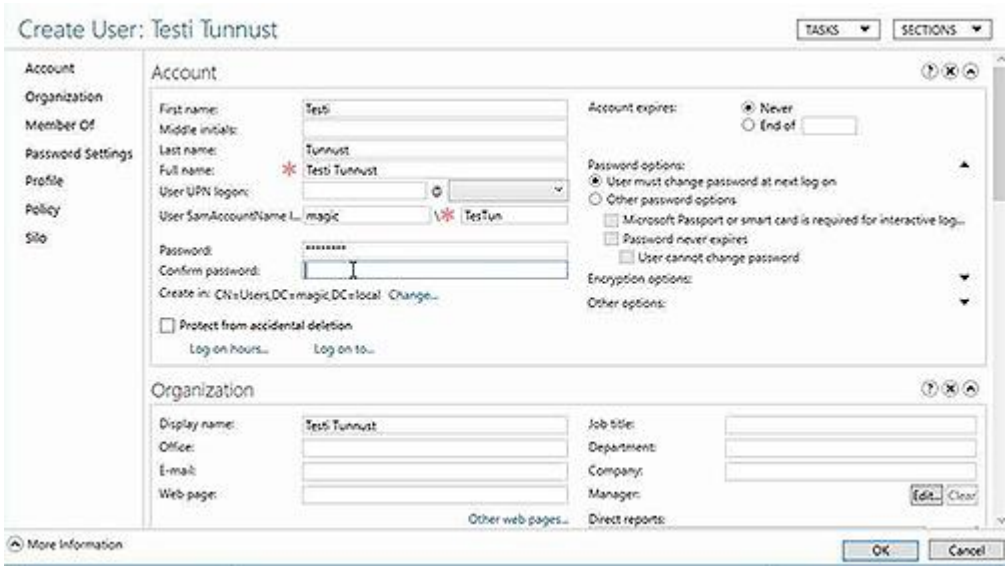


Kuva 31: Uusi käyttäjä luotu komentosarjalla

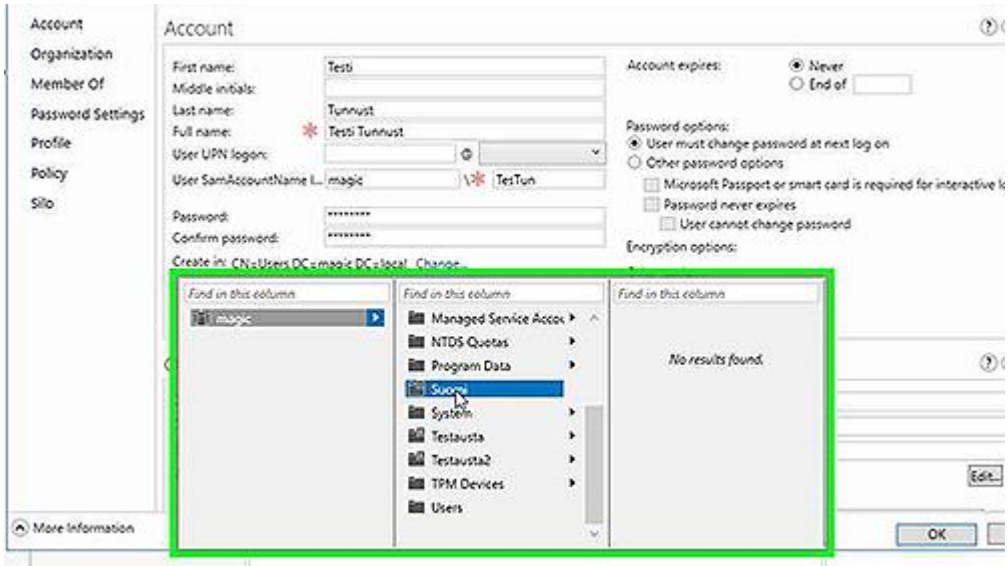
Vertailun vuoksi saman asian tekeminen Active Directory Administrative Centerissä sisältää monta vaihetta. Kuvissa 32-36 käydään läpi toimenpiteet, joiden lopputuloksena on sama tulos kuin esimerkin komentosarjalla tehtynä. Lukija voi itse päätellä kuinka kauan aikaa menee kummallakin eri tavalla tehtynä, mutta omalla kohdallani ajan säästö on puolesta minuutista minuuttiin komentosarjan hyväksi. Kuukausitasolla ajan säästö voi todellakin olla merkittävä.



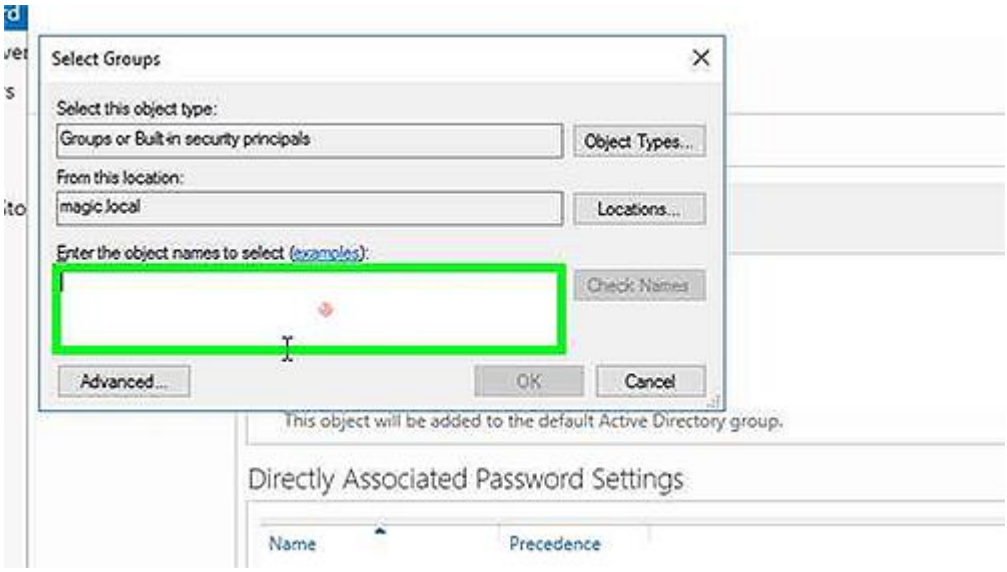
Kuva 32: Luodaan uusi käyttäjä



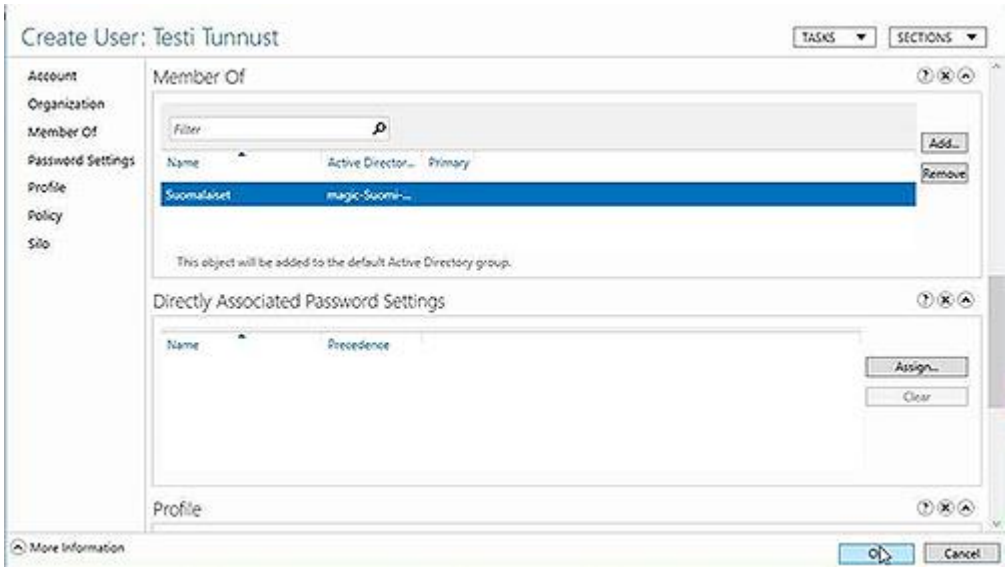
Kuva 33: Syötetään käyttäjälle nimet, käyttäjätunnus sekä salasana



Kuva 34: Lisätään käyttäjä oikeaa organisaatioyksikköön



Kuva 35: Lisätään käyttäjä oikeaan ryhmään



Kuva 36: Uusi käyttäjätunnus luodaan lopuksi painamalla OK

8 Pohdinta

Opinnäytetyön tavoitteena oli tutkia Windows PowerShelliä ja sen hyödyntämistä Windows palvelimen ja siihen asennetun aktiivihakemiston ylläpidossa. Lopputuloksena syntyikin opas erilaisiin ylläpitotehtävien suorittamiseen sekä katsaus aktiivihakemiston ja PowerShellin toimintamekanismeihin. Teoriaosassa tukeuduin laadukkaisiin ja tuoreisiin lähteisiin. Runsain esimerkein ja kuvankaappauksin esittelin miten erilaiset komennot toimivat ja miten niitä voi käyttää yhdessä toisten komentojen kanssa. Koska PowerShellin käyttömahdollisuudet ovat niin laajat, haasteena olikin päättää miten rajata opinnäytetyön aihetta. Yritin keskittyä oman kokemuksen pohjalta niihin PowerShellin komentoihin, mitä järjestelmäasiantuntija voisi tarvita päivittäin.

Mielestäni tämä raportti toimiikin mainiona lähtölaukauksena PowerShell maailmaan tutustuville. Väitän, että PowerShellin käyttötaito on nyt ja tulevaisuudessa hyvin tärkeää Windows ekosysteemissä toimiville ylläpitäjille. Microsoftin Windows-kehittäjät ovat jo vuosia panostaneet PowerShelliin ja panostus on kannattanut, he ovatkin luoneet erittäin hienon työkalun sekä kokonaisen ohjelmointiympäristön omien työkalujen tekemiseksi. Parasta PowerShellissä on sen monipuolisuus, tehokkuus ja helppokäyttöisyys. PowerShell tulee myös jokaisen markkinoilla olevan Windows-käyttöjärjestelmän mukana ja sisäänrakennetulla help-järjestelmällä sekä internetistä löytyvällä ilmaisella oppimateriaalilla jokainen halukas pääsee pitkälle PowerShellin parissa.

Kappaleen 7 komentosarjan ja graafisen käyttöliittymän vertailu uuden käyttäjän lisäämisessä toimialueeseen oli esimerkki, kuinka helppoa PowerShell komentosarjojen luominen on ja kuinka tehokkaita niistä voi saada pienelläkin vaivalla. Esimerkissä loin äärimmäisen yksinkertaisen komentosarjan, mutta sekin on jo todella käyttökelpoinen ja osoitin sen säästävään aikaa rutiinimaisessa ylläpitotehtävässä. PowerShell komentosarjoja voikin koko ajan kehittää eteenpäin lisäämällä uusia ominaisuuksia ja toimintoja. Internetissä on myös valtavasti valmiita komentosarjoja joita voi käyttää suoraan tai muokattuina lisenssiehtojen rajoissa.

Oma osaamiseni PowerShellin parissa kasvoi merkittävästi tämän opinnäytetyöprojektin myötä. Myös tiedonhankintataidot, aikataulun hallinta sekä raportointikyky on kehittynyt PowerShell-taitojen lomassa. Nyt kun PowerShellin perustoiminta on tullut tutuksi niin seuraavaksi mielenkiintoni kohdistuukin täysin PowerShellin skriptaus eli komentosarja mahdollisuuksiin.

9 Lähteet

Desmond, B., Richards, J., Allen, R. & Lowe-Norris, A.G. 2013. Active Directory. 5 painos. O'Reilly. Sebastopol.

Holmes, L. 2013. Windows PowerShell Cookbook: The Complete Guide to Scripting Microsoft's Command Shell. 3 painos. O'Reilly Media.

Jones, D. & Frame, W. 2016. Why PowerShell? The DevOps Collective, Inc.

Krause, J. 2016. Mastering Windows Server 2016. 1 painos. Packt Publishing. Birmingham.

McCabe, J. 2016. Introduction to Microsoft Windows Server 2016. Microsoft Press.

Microsoft 2017a. About Comparison Operators. Luettavissa: https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_comparison_operators?view=powershell-6 Luettu: 6.5.2018.

Microsoft 2017b. Get-Help. Luettavissa: https://docs.microsoft.com/fi-fi/powershell/module/microsoft.powershell.core/get-help?view=powershell-5.1 Luettu: 18.4.2018.

Microsoft 2017c. New Features in Windows PowerShell 5.0
. Luettavissa: https://docs.microsoft.com/en-us/powershell/scripting/whats-new/what-s-new-in-windows-powershell-50?view=powershell-6#new-features-in-windows-powershell-50 Luettu: 14.5.2018.

Microsoft 2017d. PowerShell Overview. Luettavissa: https://docs.microsoft.com/en-us/powershell/scripting/powershell-scripting Luettu: 12.4.2018.

Microsoft 2013. Windows Management Framework 4.0. Luettavissa: https://www.microsoft.com/en-us/download/details.aspx?id=40855 Luettu: 14.4.2018.

Microsoft 2009. Windows Management Framework (Windows PowerShell 2.0, WinRM 2.0 ja BITS 4.0). Luettavissa: https://support.microsoft.com/fi-fi/help/968929/windows-management-framework-windows-powershell-2-0-winrm-2-0-and-bits Luettu: 12.4.2018.

Microsoft 2008. Windows PowerShell 1.0 English Language Installation Packages for Windows Server 2003 and for Windows XP. Luettavissa: https://support.microsoft.com/fi-fi/help/926139/windows-powershell-1-0-english-language-installation-packages-for-wind Luettu: 12.4.2018.

Morimoto, R., Noel, M., Amaris, C., Shapiro, J., Droubi, O., Abbate, A. & Yardeni, G. 2017. Windows Server 2016: Unleashed. Sams.

Newman, L.H. 2017. Microsoft's Bid to Save PowerShell From Hackers Starts To Pay Off. Luettavissa: https://www.wired.com/story/microsoft-powershell-security/ Luettu: 8.5.2018.

Payette, B. & Siddaway, R. 2018. Windows PowerShell in action. Manning Publications.

Svidergol, B. & Allen, R. 2013. Active Directory Cookbook. 4 painos. O'Reilly. Sebastopol.

Wilson, E. 2014. Windows PowerShell Best Practices. Microsoft Press. Boston, USA.